Sun's Transparent Development

JAVAPro™

www.javapro.com A Fawcette Technical Publication 2006 Vol. 10 No. 1

AJAX and JSF

Build Web Apps with Pizzazz

Put Open Source into Your Infrastructure

JSTL's Faster Way to Flexible Web Applications

Speed or Quality? Decide Your Coding Priority

Empower a Plug-In to Select and Load Drivers

Adapt Disparate Objects to a Common Framework

Enterprise Architect Summit 2006

Strategies and Best Practices for the Real World

Enterprise Architect Summit returns to Florida in May for three informative days of keynotes, workshops, and breakout sessions led by experts in the enterprise architecture field. Arm your business to respond to emerging IT challenges – register today.

The Ritz-Carlton Key Biscayne Resort, Florida

May 15-17, 2006

Event Highlights:

- Microsoft on Moving Beyond SOA
- McKinsey & Company on Real-World Architecture Issues
- Intel Corporation on Enterprise Security Architecture
- Expert Panel on Modeling
- And much more...

Register by April 19 and SAVE $200

Call 800-848-5523 today

or visit us online at: **www.enterprise-architect.net/summit**

Microsoft metallect FTP FAWCETTE TECHNICAL PUBLICATIONS

Take Your Business to the Next Level-

View the list of sessions, workshops, and events planned for May

Monday, May 15

8 a.m.-6 p.m.	**Registration**	
8 a.m.-3 p.m.	Enterprise Architect Classic Golf Tournament	
	Workshops	
1-5 p.m.	**Best Practices:** Bullet Proofing Web Services	**Strategy:** IT Drivers for Business Process Management
6-8 p.m.	Welcome Reception	

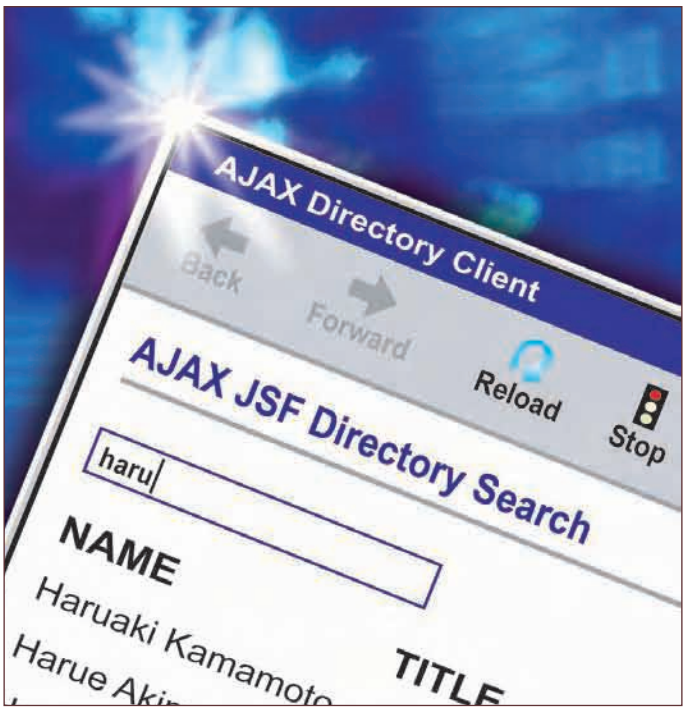
Tuesday, May 16

9 a.m.	**Keynote: Architecture on the Edge: Moving Beyond SOA**	
	Best Practices	**Strategy**
10:30 a.m.	Putting the A in SOA	Joining Enterprises with the Global SOA
11:45 a.m.	Database Connectivity in the Midst of Infrastructure Diversity	The Enterprise Architecture Office and the Ever-Increasing Organizational Need
12:45 p.m.	Lunch	
2 p.m.	**Keynote: Solving Real-World IT Architecture Issues**	
3:30 p.m.	The Rocky Road to Compliance	*Panel:* Real Solutions for Dynamic, Agile Enterprises
4:45 p.m.	Building an Agile Enterprise	Metadata: The Key to Understanding IT
6 p.m.	Exhibitor Reception	

Wednesday, May 17

9 a.m.	**Keynote**	
	Best Practices	**Strategy**
10:30 a.m.	Model Driven Software Engineering	Use a Services Networking Approach to Build a Scalable and Extendable SOA
11:45 a.m.	*Panel:* Perspectives on Architecture Modeling	The Data-Centric Enterprise: A Blueprint for Enterprise Architecture
12:45 p.m.	Lunch	
2 p.m.	Managing Dependencies Across the Architecture	Selecting SOA Infrastructure
3:15 p.m.	Minimize Business-Disruption Risk and Cost Through Architectural Modeling	SOA Operations and Governance to Move Applications to the Network Architecture
4:30 p.m.	Build an Enterprise Security Architecture	Strategic Management of SOA in the Enterprise

JAVA Pro™



COVER STORY

FEATURES

DEPARTMENTS

COLUMNS

Java Pro (ISSN: 1096-4495) is published quarterly by Fawcette Technical Publications Inc., 2600 South El Camino Real, Suite 300 San Mateo, CA USA, 94403. Tel. (650) 378-7100; fax (650) 570-6307. **POSTMASTER:** Send address changes to ***Java Pro***, P.O. Box 3485, Northbrook, IL 60065-9819. Periodicals Postage Paid at San Mateo, CA and at additional mailing offices. **Canadian CPM: 40013204; Customer: 2418150; Return Address: 4960-2 Walker Road, Windsor, ON 49A 6J3, Canada. ISSN: 1096-4495.**

PUBLISHER, Jeff Hadfield

EDITORIAL
EDITOR, Terrence O'Donnell
TECHNICAL EDITOR, Daniel F. Savarese

CONTRIBUTORS
Kevin Jones, Daniel F. Savarese, and Peter Varhol

ART & PRODUCTION
VICE PRESIDENT, ART & PRODUCTION, Michael Hollister
SENIOR ART DIRECTOR, Bruce Gardner
PRODUCTION MANAGER,
Kathleen Sweeney Cygnarowicz
ART DIRECTOR, Brian Rogers
SENIOR INTERACTIVE ART DIRECTOR/WEB PRODUCER,
Lyndon Lloyd
ASSOCIATE WEB PRODUCER, Shane Lee

ADVERTISING SALES
AD DIRECTOR, VSM AND JAVA PRO, Kevin White
EXECUTIVE ASSISTANT TO THE VICE PRESIDENT, PUBLISHING,
Susan LaCroix

CIRCULATION
SENIOR CIRCULATION DIRECTOR, Karen Koenen
CONFERENCES ASSOCIATE, Gerry Guzman

MARKETING
MARKETING MANAGER, Susan Ogren
SENIOR DESIGNER, Margaret Horoszko

CONFERENCES
VICE PRESIDENT, Tim Smith
SALES OPERATIONS MANAGER, David Seymour
CONFERENCE OPERATIONS PLANNER, Will Hansen
MARKETING EDITORIAL PLANNER, Katie McGillivray
EXHIBIT SALES MANAGER, Tina Fontenot

CUSTOMER SERVICE
CUSTOMER SERVICE REPRESENTATIVE, Jose Porcell

OPERATIONS
EXECUTIVE VICE PRESIDENT/CHIEF FINANCIAL OFFICER,
John Sutton
SYSTEM ADMINISTRATOR, Tin Cao
DIRECTOR OF FINANCE, Darlyn Phillips
ACCOUNTS PAYABLE ACCOUNTANT, Elena Ostrovsky
STAFF ACCOUNTANT, Betty Tsang-Hwah Wu
ACCOUNTS RECEIVABLE, Iain Niellands
HUMAN RESOURCES MANAGER, Pam Davis

FTPONLINE
MANAGING EDITOR/BUSINESS UNIT MANAGER,
Nina Goldschlager
ADVERTISING DIRECTOR, Roy Kops
PROJECT MANAGER, Fred Perry
SENIOR EDITOR, Terrence O'Donnell
ASSOCIATE EDITOR, Lauren Dresnick
EASTERN REGIONAL SALES MANAGER, Dennis Leavey
WESTERN REGIONAL SALES MANAGER, Lisa Sidlow

FAWCETTE TECHNICAL PUBLICATIONS
PRESIDENT, James E. Fawcette
VICE PRESIDENT, CHIEF INFORMATION OFFICER, Aaron Weule
VICE PRESIDENT, PUBLISHING, Jeff Hadfield
CORPORATE COUNSEL, Wilson, Sonsini,
Goodrich & Rosati

JAVAPro online

Visit *Java Pro* online for this extended content and more.

The ONLY place on the Web for *Java Pro* content, code, and community is www.javapro.com

FTPOnline Blogs

Check out the FTPOnline blog page to get insights from Jeff Hadfield, vice president, publishing, Terry O'Donnell, *Java Pro* editor, and other FTP editors sounding off on IT and development issues.
www.ftponline.com/weblogger/

Tale of Two Cities: EclipseCon & MIX06

Locator+ code: JF_060324
http://www.ftponline.com/weblogger/forum.aspx?ID=1&DATE=3/24/2006
by Jim Fawcette
Industry figures speaking at EclipseCon for open source Java developers and MIX06, Microsoft's outreach to the Web 2.0 crowd, according to Jim Fawcette, apparently agree that Vista will create major market opportunities for everyone. Find out what the big differences were between these conferences beyond being separated by 530 miles.

More Online Exclusives

Enforce Application Architecture

Locator+ code: JC0306PV_T
by Peter Varhol
We've all spent hours with code that needs reliable enhancement with new features. See how DSM tracks dependencies from the initial design to reduce the impact of changes over time.

Keeping Up Appearances

Locator+ Code: JC0306JS_T
by Terrence O'Donnell
Ever wonder what makes certain products "blue chip hits"? How do you get customers to fall in love with your newly developed applications? Relive some of the glamour, glitz, and insights from Joel Spolsky, Fog Creek Software's CEO, as he tackled these questions in his keynote at EclipseCon 2006.

What Are Locator+ Codes?

Locator+ codes give you instant access to a feature on *Java Pro* Online. Simply type the Locator+ code into the field in the upper-right corner of the page, and click on the "go" button.

JAVA INSIGHT NEWSLETTER SIGN-UP

www.javapro.com
Every week, the *Java Insight* e-mail newsletter brings you up-to-date news, technical information, opinions, interviews, and analysis on topics and technologies such as J2EE, middleware, and application servers; J2ME, devices, and embedded development; data access; servlets and JSP; Web services; and/or XML. Sign up for *free* at www.javapro.com.

Units of Measure

by Terrence **O'DONNELL**

Fawcette Technical Publications, the publisher of this magazine, is based in the San Francisco Bay Area, and this year for some reason—El Niño, global warming... take your pick—the jet stream that controls West Coast weather patterns has managed to lose its way and confuse Northern California for the Pacific Northwest. So much so that during the last few days of March the Bay Area had surpassed a reportedly 102-year-old record for most days (25) in the month of March where there were measurable amounts of precipitation. This year's EclipseCon conference in Santa Clara managed to somehow squeeze itself into some of the few dry days we had that month, bringing along some remarkable numbers of its own in terms of membership: 130 members, 630 committers, and 61 projects.

At a press conference during the event, Mike Milinkovich, executive director of the Eclipse Foundation, provided an overview of its status, noting that the Eclipse platform is now five years old, mature, and stable. In assessing Eclipse's success, Milinkovich said "we measure [it] to a large degree in terms of ISV adoption: who is using Eclipse to build products? And one of the things that distinguish Eclipse from many of the other open source communities is that we are very explicit and conscious about wanting to see the commercial adoption of the technologies we're building at Eclipse."

Milinkovich then outlined the distinct *pillars* of the platform, each supporting an ecosystem, that like any organic entity can contract or expand in number over time through growth and change. The Eclipse story demonstrates an area in the industry where there is a lot of interesting innovation taking place, and Milinkovich has some perspectives about that topic that he shares in a guest opinion piece for the Public Static column.

Speaking of open source and numbers, there have been some interesting developments taking place at Sun Microsystems over the past year that are noteworthy for the community of developers and other IT professionals. After the release of their Tiger project (Java SE 5) in 2004, Sun's engineering leadership had an epiphany that led to a new policy of transparency in which development of selected platforms, tools, and projects would be made available to outside developers, who were invited to provide their feedback. One of many examples that resulted from this initiative is that Sun has been posting source code and binaries during development of its Mustang project (Java SE 6), instead of *after* the release as has been the norm historically for other projects. Sun reports that the benefits to the organization, its own engineers and developers, and the outside developer community have been plentiful, and the droves of developers registering at the SDN site support this contention. If you want to see some of the details of Sun's transparency model, take a look at the In Brief column.

Although perhaps not as quantifiable, but no less significant, is this issue's good mix of featured topics that we've all been hearing quite a bit about lately. Our cover feature provides a cool demonstration of using AJAX-enabled, JavaServer Faces (JSF)-component technology to fully realize the richer and robust end-user experience of Web applications through creative exploitation of client-side JavaScript. On a more granular level, though scripting languages are all the rage, you can read about using the JavaServer Standard Tag Library (JSTL) for a faster approach that promotes more flexibility in building Web applications as well as a means to build them more quickly.

You will also find in these pages a wide-ranging discussion of open source and how the extensive availability of source code and software that provides platforms and middleware for building and deploying applications is benefiting enterprises. However, there are other ramifications to adopting the open source model, and this discussion offers additional insights for organizations considering a shift to this alternative business model for developing software.

Of course, there is also the expertise and perspectives of our regular columnists to round out this issue, and you'll find that each provides a unique perspective on the development of tools that spans educational and training considerations for future IT professionals, a specialized task for a generic Eclipse plug-in, and how even a bit of *trickery* can overcome recurring challenges when building custom tools.

It all measures up to an informative read. JP

Terrence O'Donnell, Editor
todonnell@fawcette.com

Developer-Friendly Transparency

Sun Microsystems is realizing the benefits of opening up development resources to outside developers

Sun Microsystems announced recently two new programs aimed specifically at developers that offer the latest services in their much broader efforts to make development of their platforms, tools, and programs transparent to the outside developer community. The Sun Developer Expert Assistance Program (http://developer.sun.com/services/expertassistance/) is a per-incident support offering that is now available. For a fee of $99 per incident, a developer can get a guaranteed response to a coding question within 24 hours. According to Jean Elliott, director of developer marketing at Sun, the company has in the past offered similar support in a variety of channels, from active forums on the Sun Developer Network (SDN) to sophisticated professional services contracts and everything in between.

This new program, however, gives developers looking for inexpensive per-incident support the option of buying a guaranteed response time, the answer or answers to their coding question(s), or a commitment to track down the answer, depending on the complexity of the issue. Elliott said that although developers have found great support from Sun engineers and community engineers through forums, there's no guarantee of a response through those support mechanisms. For each registered incident, the support will continue until the developer is satisfied. "This program is good for help on diagnostics, getting a sanity check on your approach to the code, finding workarounds, getting guidance for best practices, and also it's a way developers can find pointers to various sample applications or helpful documentation to help them address their issues," Elliott said.

The new program's offering coincides with an overhauled SDN Web site page (http://developers.sun.com/services/) that consolidates and better organizes all of the information on the services that are available to developers. Developers can go to the site and choose an offering that's appropriate for their price point and the complexity of what they're trying to address.

The launch of the Sun Community Champions Program is the second new offering announced recently by Sun. This program, which according to Elliott is a natural extension of the Java Champions Project on the Java.net site, offers developers doing interesting projects with Sun's software products and technologies beyond just Java a chance to be recognized for their work. The recognition is not only a way for Sun to thank them, but it gives developers an opportunity to serve as role models for other developers.

Two-Way Feedback

These two programs are the recent results of a larger, nearly yearlong effort to coordinate Sun's platforms, programs, tools, and community and provide developers the transparency necessary for them to benefit from Sun's development as well as provide feedback on that development. The primary objective, according to Elliott, is to ensure there are interesting software applications that are at least consistent, if not optimized for, Sun's software and hardware architectures. To achieve that goal Sun needs to attract developers to its tools, technologies, platforms, and programs to inspire them to create those applications. The effort also culminated from Sun's leadership realizing that in the past there have been some inconsistencies, whether there was incompatible tools support for Sun's platforms; stale content; or services, documentation, or training not being made available when developers really needed it.

Sun's executives also realized that the developer community was starting to really do some interesting things in software development, and that the company perhaps wasn't listening as closely as it should have been prior to when Java SE 5.0 (Tiger) was released in September 2004, Elliot said. Following that release, Sun began listening closer to the community and several themes emerged. Elliott said one theme that proved to be the biggest wake up call for the company's engineering leadership was hearing feedback about how the new release exhibited the hard work put into the platform, but that the outside community had no idea what Sun was doing, and the community desired to know a lot more of what was going on.

Sun also couldn't ignore the allure of open source. Elliott said Sun spent a lot of time "teasing out" the properties of the open source model that were interesting because open source was seen as a means to a variety of ends, depending on who the customer is. "The transparent development model is an interesting property of the open source model," Elliott said. "Developers can know what's happening, what's being done in the code, what the status is, how it's being implemented, who's involved, and understand the rationale for the development decisions and the trade-offs that were made."

Sun also recognized flexibility of licensing as another attractive property of open source that can help drive innovation or reduce a deploying customer's anxiety over potentially being stuck waiting for a

vendor's next release before deployment of their application. Of course, open source also gives some developers the ability to contribute, which Elliott admits was another significant factor in Sun taking a serious look at the development model. Elliott said that the community was clearly calling for Sun to "engage more broadly with the community [because] this is how a lot of interesting software is being developed today." According to Elliott, the community's appeal was clear: "make it a two-way conversation, listen to the community, tell us what you are doing, help us help you, and show us where you're going so that we can all benefit together."

Culture Shift

Sun's commitment to its new policy toward the developer community is pervasive throughout the organization, and it means that Sun has redefined its engineers' job descriptions. Sun engineers must spend 20 percent of their time performing functions that don't involve writing code but instead involve helping educate, inspire, or help the community. These noncoding tasks could be speaking engagements at the company sponsored Tech Days presentations, spending time with Java users' groups or at customer sites, or other tasks such as helping to clean up some of the dated Web site content, writing sample code, or reviewing training and certification class materials. Elliott said that this redefinition of the engineer's role demonstrates a major cultural shift for Sun's engineers, but it is producing results that developers really do value.

These efforts are paying off well this year, according to Elliott. Sun started its current fiscal year with just over one million registered developers at SDN, and internally the company had the ambitious goal to double that registration number by the end of the year. Yet the company is on track to exceed that goal, having had approximately 1.75 million registered developers at the beginning of spring 2006, which if continued at an accelerated rate through the rest of the year, amounts to roughly two developers per minute joining the SDN site. Visit the SDN site for more information.

Developer Tools

Actuate BIRT 2.0

Acutate's Business Intelligence and Reporting Technology (BIRT) 2.0 is an Eclipse-based, open source, reporting tool that gives you the ability to tackle complex reporting needs in Web-based and other applications. The tool provides out-of-the-box integration for commercial data sources and functionality that mirrors the Eclipse BIRT 2.0 project, which is the Eclipse Public License (EPL) equivalent of the Actuate BIRT 2.0 tool. Actuate-supported licensing, maintenance, and professional services are available. "With BIRT we're going to follow through the model that Eclipse did to build an ecosystem," said Mike Thoma, vice president of product marketing, Actuate Corporation. Thoma said that Actuate added a report component library, and BIRT evolved from a 1.0 product to a 2.0 product in terms of the reporting engine, which provides for doing very large reports but has the persistence without having to touch the database. Actuate also lowered the bar from the Java-developer level down to a level where you don't have to be a technical expert to do new report types and interactive chart types. However, for the senior Java developers there is Java as a scripting language and the product takes advantage of the Eclipse IDE for debugging. Visit Actuate's Web site for more information and pricing.

Actuate Corporation
650-837-4837
Fax: 509-696-6964
www.actuate.com

Eclipse Rich Client Platform

The Eclipse Foundation announced four open source initiatives at its EclipseCon 2006 event that extend the Rich Client Platform (RCP). The Eclipse RCP is an integration platform on which rich client applications are built and deployed. The primary benefit of the RCP, according to Eclipse Foundation release notes, is that software vendors and enterprises can deploy applications on several operating systems without being tied to a single operating system. The four initiatives provide application frameworks for specific functionality that developers can put into Eclipse RCP-based applications. The Eclipse Data Tools Project (DTP) is a new release that provides a set of tools and frameworks for creating data-centric applications on the RCP and allows for defining and managing multiple datasource connections. The Eclipse Communication Framework (ECF) project provides for creating RCP applications that use client-server and peer-to-peer messaging and communications, and it introduces protocol-neutral Voice over Internet Protocol (VoIP). Apogée, or the Enterprise Content Management Project, is a new project proposed by Nuxeo for creating a global client platform on which enterprise content management (ECM) applications can be built. It provides for building RCP-based applications that integrate document management, business processes, collaboration, and business forms, and it will be able to connect with existing ECM infrastructures. The Enterprise Component Framework Project is a recent proposal by Exadel for extending the Equinox framework to make it possible to build and use Eclipse components in server applications. Visit the Eclipse Foundation's Web site for more information.

Eclipse Foundation
www.eclipse.org

Exadel Visual Component Platform

Exadel Visual Component Platform (EVCP) 1.0 is a rich component framework for developing business applications based on JavaServer Faces (JSF). The platform offers AJAX support and provides a high level of abstraction for developing component-based, rich Web applications and hides the complexities of underlying technologies. Exadel claims that EVCP has a revolutionary design that represents a paradigm shift from hand-crafted development to an industrial way of composing applications from components. Exadel provides software, services, and support that allow companies to create mission-critical, business applications based on open source and Java technologies. The EVCP provides a robust visual library of AJAX components and extends JSF development environments by offer-

ing components, AJAX-enabled components, and skin-enabled components. "For application developers and Java developers they have to use AJAX, no question, and they have to make sure the application is changeable; we provide a very exciting technology that is very important for developers," said Fima Katz, founder, president, and CEO of Exadel. In addition to AJAX support, important features include access to common UI components, easy customization of the UI's look and feel through themes and skins, full support for the JSF life cycle using AJAX, an advanced 3-D UI, and drag-and-drop functionality for Web applications. Visit Exadel's Web site for more information and pricing options.

Exadel Inc.
925-363-9510; 888-439-2335
Fax: 925-363-9509
www.exadel.com

ULC Visual Editor 5.0

The new release of Canoo's ULC Visual Editor 5.0 for Eclipse 3.1 now runs with Eclipse 3.1 and the Eclipse Visual Editor 1.1. Porting to Eclipse 3.1 has greatly improved the overall performance and stability of the tool, and the editor lets you visually compose Rich Internet Applications (RIAs). ULC Visual Editor provides a drag-and-drop UI designer for UltraLightClient, a Java library for RIA development. The library uses Swing on the client, standard communication protocols set by the Java EE container, and standard life-cycle management on the server. The principal design is based on the half-object and protocol design pattern that when applied to Swing offers the Swing API in a server-side programming model to provide rich user interfaces in a Web architecture. UltraLightClient is a pure, Java-based alternative to other RIA technologies like AJAX and Macromedia Flex that leverages UltraLightClient's code maintainability, reliability, scalability, performance, and security. UltraLightClient-based applications are preferable for corporate intranet applications or B2B applications running over the Internet. New features of the ULC Visual Editor include improved startup performance when opening a visual class, support for ULCFiller, a new class wizard to generate ULC applications, and limited copy-and-paste functionality. Visit Canoo's Web site for more information and pricing.

Canoo
+41 (0)61 228 94 44
Fax: +41 (0)61 228 94 49
www.canoo.com

Database

InPowerForms RCP

InPowerSoft Corporation's recent release of InPowerForms RCP is a database-driven application development platform that was ported from the original Java Swing implementation to the Eclipse platform. InPowerForms RCP is a rich-client development platform that provides Oracle Forms-like functionality, and is implemented in Java Eclipse RCP/JBoss Hibernate. It provides a level of abstraction that allows for any database-driven application to be developed significantly faster than any other technology stack. InPowerForms RCP is database neutral, and it provides increased speed, reliability, consistency, and robustness for database-driven application development. InPowerSoft also provides a simple human resource application, InPowerHR, that is available for download. It was built on InPowerForms RCP and offers a way to learn about using the framework. Visit the InPowerSoft Web site for additional information.

InPowerSoft Corporation
503-343-0277
www.inpowersoft.com

Compiled by Terrence O'Donnell

HOW TO REACH US

Editorial Offices: Fawcette Technical Publications, 2600 South El Camino Real, Suite 300, San Mateo, CA 94403; Phone: (650) 378-7100; Fax: (650) 570-6307; Web: www.javapro.com; e-mail: java-pro@fawcette.com.

Article Submission: To submit articles for publication please contact Terrence O'Donnell, Editor, todonnell@fawcette.com. Download the Author Guidelines at www.ftponline.com/javapro/code/12dec01/author_guide.zip.

Product Announcements: To submit announcements for new products or updates to existing products, please e-mail press releases to Terrence O'Donnell, Editor, todonnell@fawcette.com.

Reprints and Permissions: For all *Java Pro* editorial and advertising reprints contact Serv U Reprints, 101 Rhoades Way, Folsom, CA 95630; Phone: (916) 983-6562; Fax: (916) 983-6762.

To Quote from an Article: Please contact Susan LaCroix, 2600 South El Camino Real, Suite 300, San Mateo, CA 94403; Phone: (650) 833-7118; or e-mail: slacroix@fawcette.com. Specify the issue date and title of the article, the portion you would like to quote, and the purpose.

Photocopy Rights: Permission to photocopy for internal or personal use may be granted by Fawcette Technical Publications. Please contact Susan LaCroix at slacroix@fawcette.com for more information.

Customer Service and Subscription Information: For subscription orders, inquiries, or address changes please contact Customer Service, *Java Pro*, P.O. Box 3485, Northbrook, IL 60065-9819; Phone: (866) 387-5776; Fax: (847) 291-4816; for international inquiries call (847) 559-7309; or e-mail jva@omeda.com. Foreign and Canadian orders must be payable in U.S. dollars, plus postage. The surface rate to Canada and Mexico is $52.97 per year. For all other countries the air mail rate is $78.97 per year.

Back Issues: To order *Java Pro* back issues, call (650) 378-7100 or (800) 848-5523 and ask for Customer Service. Back issues cost $10. Additional postage will be charged to deliveries outside the USA.

From the Editors of *Java Pro*:

JAVAinsight

FREE Weekly E-Mail News

How Much Java™ Do YOU Want?

Get the best in Java news, resources, how-to tips and more delivered to your inbox every week—FREE. With *Java Insight*, it's easy to keep up on the latest Java news and information.

Delivered every single week to your inbox, it gives you regular, spirited coverage of hot topics like:

- The latest in J2EE technology
- Using Java to develop Web services
- Creating Web apps with JSP
- Extending Java to embedded and wireless systems and devices
- And much, much more!

Sign up online at: **www.javapro.com**

And while you're there, check out the complete FTPOnline network of technical sites for IT development professionals.

FTPOnline

The AJAX Approach to Richer Interfaces

Apply AJAX-enabled, JSF-component technology to simplify robust, interactive Web applications

by Chris **SCHALK**

In the relatively short history of the Web there has been a constant evolution of technologies ranging from the Common Gateway Interface (CGI), Perl, Active Server Pages (ASP), JavaServer Pages (JSP), to JavaServer Faces (JSF). However, what has remained largely the same throughout the Web's history until recently has been the familiar notion of building applications that constantly refresh complete pages.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0601 Download all the code for this issue.

Read More

JP0601CS_T Read this article online.

JP0511PB_T Read the related article "The Year of AJAX" by Kito D. Mann.

JP0505MD_T Read the related article "Implement a Graphical JSF Component" by Mark Durocher.

JP0401CS_T Read the related article "Developing Web Interfaces with JSF" by Chris Schalk.

A new class of Web applications is now beginning to emerge that offers a familiar *rich client* behavior that was predominant just before the Web became popular. The development of newer, richer Web applications, often referred to as rich Internet applications (RIA), provides the user with a much more vivid experience that is similar to desktop applications. Gone is the notion of constantly having to refresh the entire page for each transaction. The key to providing this much richer end-user experience over the Web is the clever usage of client-side JavaScript and/or DHTML using an asynchronous JavaScript and XML (AJAX) approach.

AJAX by itself provides a superior end-user experience, but it relies heavily on advanced JavaScript coding. However, when coupled with JSF, this complexity can be reduced drastically. AJAX allows a Web client to make requests in an asynchronous manner to the server, independent of the end user submitting the entire page. In addition to greatly improving the user interface interaction it also allows for a more sophisticated event model where interactions are based on specific UI events as opposed to a simplistic HTTP GET or POST event where the entire page is refreshed.

Is AJAX new? Not really. Remote JavaScript, of which AJAX is one example, has garnered the most attention as of late and provides the ability to interact with a server by using XML data. AJAX is possible because the leading browsers now offer *objects* that can make independent XML HTTP requests. Microsoft Internet Explorer 5 and later offers an XMLHTTP object, while Mozilla-based browsers provide an XMLHttpRequest object. Both objects offer essentially the same ability to request XML data from a server and process the response data in a similar fashion.

The required server-side AJAX technology, which interacts with the client-side JavaScript, can be built using any Web technology that can dynamically generate XML (or any markup) ranging from PHP to Java servlets.

A JSF Solution

As you might guess, creating client-side JavaScript code to communicate asynchronously with a server can often be a very complex challenge, especially when working with an intricate user interface. Fortunately, this challenge is where JSF provides a solution and makes AJAX very easy to use by encapsulating complex AJAX JavaScript code into the JSF-component technology. Before we review examples of AJAX-enabled JSF components, it is useful to understand the core AJAX architecture involved in an AJAX client-server transaction.

As mentioned previously, AJAX is possible, providing the two core technologies are present: a JavaScript-enabled browser that supports either the XMLHTTP or XMLHttpRequest object and an HTTP server technology that can respond in XML. Since today's popular browsers support JavaScript and the necessary XML HTTP request objects, and almost any Web server technology can generate XML (or any markup), the core AJAX technology is widely available. The architecture for a pure AJAX application is very straightforward (see Figure 1).

As you can see in Figure 1, an AJAX application in its simplest form is essentially a standard HTML user interface with

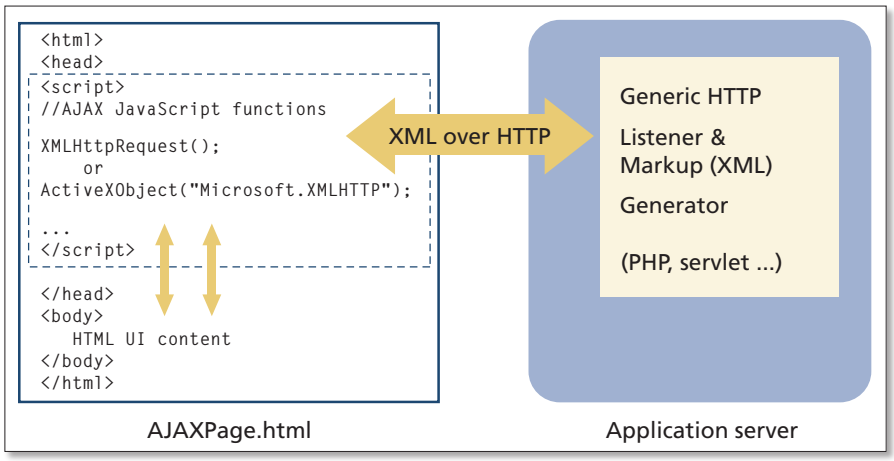


Figure 1 | Core AJAX The core AJAX architecture for a pure AJAX application is straightforward.

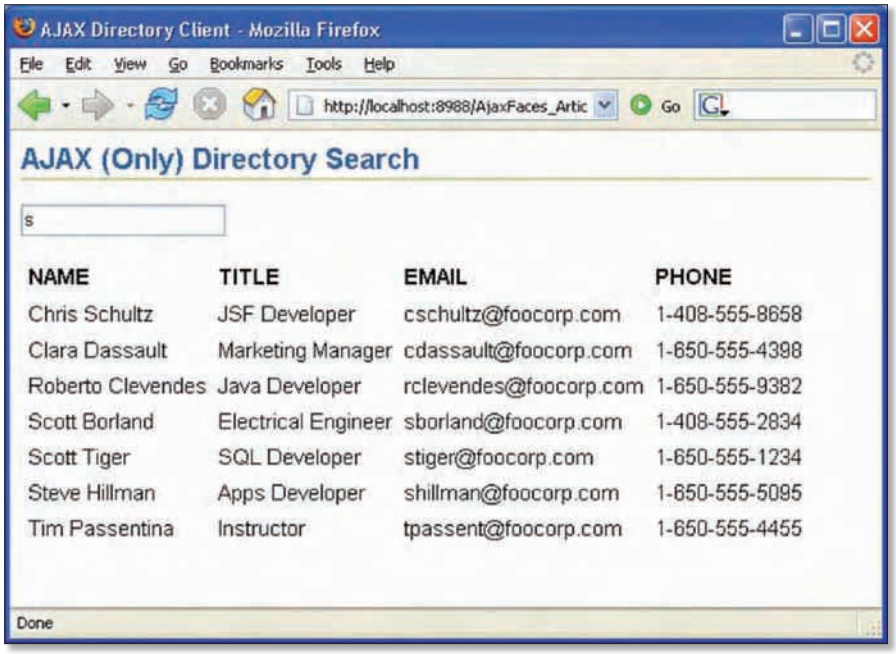


Figure 2 | Smart Field The AJAX DirectorySearch application presents a single input text field that displays a list of corresponding matches from a directory when a user begins typing characters into the field.

JavaScript functions to interact with any type of HTTP server technology that can handle the request and respond dynamically in XML. The key elements of a core AJAX application are an HTML page and a server-side Web technology that can process HTTP requests (such as Java servlets, PHP, and so on) and respond in XML markup. The HTML page contains user interface (UI) elements that interact with AJAX JavaScript functions and JavaScript functions that interact with an AJAX server.

Reviewing the key elements in a more realistic scenario involves an HTML UI with elements such as an input field, a button, or anything that can be linked to JavaScript. For example, a button could fire a JavaScript function when pressed, or for even more subtle usage an input field could fire a JavaScript function as the user types into the field. This functionality is accomplished by setting the onkeyup attribute of the input field to a JavaScript function that processes the data in the input field. For example, the input field searchField will call the JavaScript function lookup() when an onkeyup event occurs (that is, during typing):

```
<INPUT TYPE="TEXT" ID=
  "SEARCHFIELD" SIZE=
  "20" ONKEYUP="LOOKUP(
  'SEARCHFIELD');">
```

In addition to responding to UI interactions (like typing), AJAX JavaScript functions can operate independently on their own timers. (An AJAX autosave feature can be implemented using this approach.)

Now that we've reviewed how the AJAX JavaScript code can be invoked, let's review the JavaScript code that can issue an XML HTTP request. Recalling that both major browser families support remote JavaScript in a similar fashion, some code can be added to a page to first check for its support and then instantiate the appropriate object:

```
IF (WINDOW.XMLHTTPREQUEST) {
  REQ = NEW XMLHTTPREQUEST();
}
ELSE IF (WINDOW.ACTIVEXOBJECT) {
  REQ = NEW ACTIVEXOBJECT(
  "MICROSOFT.XMLHTTP");
}
```

Once instantiated, it can be operated on in exactly the same manner. To initialize a connection to a server, use the open method:

```
REQ.OPEN("GET", URL, TRUE);
```

The first argument is the HTTP method (GET or POST). The second argument is the server's URL (or form action if using a POST). When true, the third argument denotes whether the call should be made asynchronously (hence, the "A" in AJAX). An asynchronous call means that the browser can continue doing other things while the request is being fulfilled. A false value in the open method denotes a non-asynchronous or serial processing, which is not recommended because your browser will cease operations until the response has been returned.

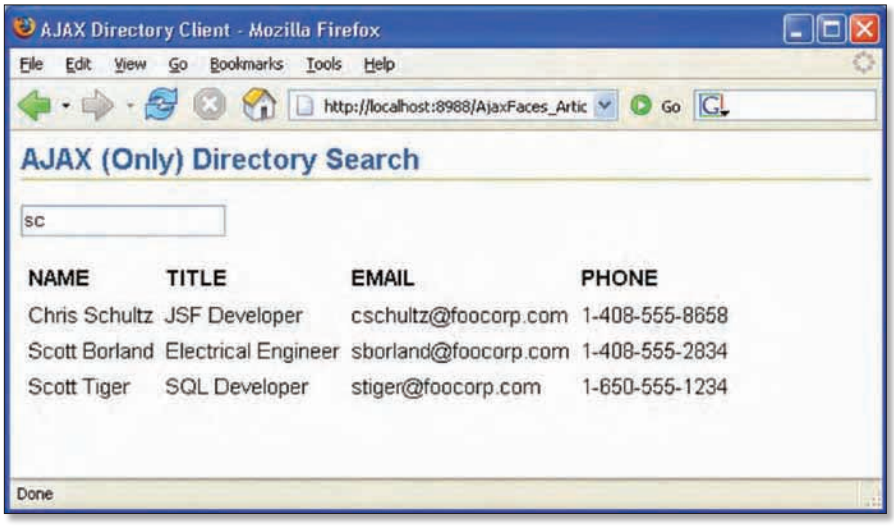


Figure 3 | No Refresh Required When a desired match from the input displays, the page updates itself with the most correct matches without requiring a traditional page submission and a complete refresh cycle.

Making the Connection

After using "open" to initialize a connection, an onreadystatechange call is made (only for asynchronous calls). This call registers a callback function, which will be invoked once the request is complete:

```
REQ.ONREADYSTATECHANGE =
  PROCESSXMLRESPONSE;
```

Listing 1 Initiate the AJAX Call

```
<script type="text/javascript">

var xmlreq;
var writeloc;

function lookup(field) {
  writeloc = field + ":table";
  var searchField = document.getElementById(field);
  var url = "ajaxdirectoryservice?input=" +
    searchField.value;

  if (window.XMLHttpRequest) {
    xmlreq = new XMLHttpRequest();
  }
  else if (window.ActiveXObject) {
    xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
  }

  xmlreq.open("GET", url, true);
  xmlreq.onreadystatechange = processXMLResponse;
  xmlreq.send(null);
}
```

The purpose of the JavaScript lookup() function is to initiate the AJAX call with a URL that is the (mapped) name of the AJAX servlet.

Listing 2 Render a Table

```
function renderTable()
  {
    xmlDoc =  xmlreq.responseXML;
    var elements = xmlDoc.getElementsByTagName(
      'employee');
    var table = document.createElement('table');
    table.setAttribute('cellPadding',3);
    table.setAttribute('border',0);
    var tbody = document.createElement('tbody');
    table.appendChild(tbody);
    var h_row = document.createElement('tr');

    for (i=0;i<elements[0].childNodes.length;i++) {
      if (elements[0].childNodes[i].nodeType != 1)
        continue;
      var t_header = document.createElement('th');
      var headerData = document.createTextNode(
        elements[0].childNodes[i].nodeName);
      t_header.appendChild(headerData);
      h_row.appendChild(t_header);
    }
    tbody.appendChild(h_row);

    for (i=0;i<elements.length;i++) {
      var t_row = document.createElement('tr');
      for (j=0;j<elements[i].childNodes.length;j++) {
        if (elements[i].childNodes[j].nodeType != 1)
          continue;
        var td = document.createElement('td');
        var tdData = document.createTextNode(
          elements[i].childNodes[j].
          firstChild.nodeValue);
        td.appendChild(tdData);
        t_row.appendChild(td);
      }
      tbody.appendChild(t_row);
    }

    // Clear previous table
    var element = document.getElementById(writeloc);
    while(element.hasChildNodes())
      element.removeChild(element.firstChild);

    // Append new table
    document.getElementById(writeloc).appendChild(
      table);
}
```

The JavaScript renderTable() function iterates through the XML data, constructs an HTML table from it, and appends it to the original HTML page.

Listing 3 Input Process

```
package com.ajaxjsf.servlet;

public class AjaxDirectoryService
  extends HttpServlet {
  public void init(ServletConfig config) throws
    ServletException {
      super.init(config);
  }

  public void doGet(HttpServletRequest request,
              HttpServletResponse response)
              throws ServletException,
              IOException {
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");

    XmlGenerator myXmlGenerator = new XmlGenerator(
      request.getParameter("input"));

    String xmlout = myXmlGenerator.getXmlResponse();
    PrintWriter out = response.getWriter();
    out.write(xmlout);
    out.close();
  }
}
```

The AjaxDirectoryService servlet responds to the XML HTTP request and processes the incoming input parameter.

This callback function example, processXMLResponse(), processes the returned XML response and is invoked when the request is fulfilled. A callback function can also be declared inline in the onreadystatechange statement:

```
REQ.ONREADYSTATECHANGE =
  PROCESSXMLRESPONSE() {
  // PROCESS REQUEST
};
```

Oftentimes it's necessary to set a header value in the XML HTTP request object by using setRequestHeader(). For example:

```
REQ.SETREQUESTHEADER(
  "COOKIE", "SOMEKEY=TRUE");
```

Once the XML HTTP request object (req) has been fully initialized, you can use send() to initiate a call to the server:

```
REQ.SEND(NULL);
```

For GET requests, a null value or empty string "" is used. POST requests contain a string argument with form data. They also require the Content-Type to be set in the header of the request. These two lines show how to perform an AJAX POST request:

```
REQ.SETREQUESTHEADER(
  "CONTENT-TYPE",
  "APPLICATION/
  X-WWW-FORM-URLENCODED";
REQ.SEND(
  "NAME=SCOTT&EMAIL=
  STIGER@FOOCORP.COM");
```

The callback function, which is called once the request has been fulfilled, usually has some code to make sure the request has not resulted in an error. You can accomplish this reassurance by checking the readyState as well as the overall status of the HTTP request. (A readystate of 4 means the XML HTTP request is complete, and 200 means it was a success—as opposed to 404.)

```
FUNCTION PROCESSXMLRESPONSE() {
  IF (XMLREQ.READYSTATE == 4) {
    IF (XMLREQ.STATUS == 200) {
```

```
    // PROCESS THE XML RESPONSE...
    }
  }
}
```

You can use standard JavaScript DOM methods to process the XML response. For example, to extract the employee name from the incoming XML stream shown here:

```
<EMPLOYEE>
  CHRIS
</EMPLOYEE>
```

use this method:

```
VAR NAME = REQ.RESPONSEXML.
  GETELEMENTSBYTAGNAME(
  "EMPLOYEE")[0];
```

Parsing more complex XML usually involves iterating through the elements using code such as this:

```
FOR (I=0;I<ELEMENTS.LENGTH;I++)
  {
  FOR (J=0;J<ELEMENTS[I].
    CHILDNODES.LENGTH;J++) {
    VAR ELEMENTDATA =
      ELEMENTS[I].CHILDNODES[J].
      FIRSTCHILD.NODEVALUE;
  }
}
```

Be aware that the XML response obtained through the XML HTTP request object doesn't always need to be well formed and valid. The AJAX server-side component can send HTML content directly to the client. JavaScript can then retrieve the HTML content by using the req.responseText() method/property, which simply retrieves the content as a string. The HTML string text can then be used in whatever fashion to alter the page. For example, this HTML stream:

```
<H3>HELLO THERE!</H3>
  <P> THIS IS <B>HTML</B></P>
```

could be retrieved into a string using:

```
VAR HTMLCONTENT =
  REQ.RESPONSETEXT;
```

and then added to a specific HTML DIV tag:

```
DOCUMENT.GETELEMENTBYID("DIV1").
  INNERHTML += HTMLCONTENT;
```

Having stepped through the basics of an AJAX transaction, let's consider a pure AJAX application example that we can later compare to a revised version of the same example implemented with JSF technology.

> AJAX by itself provides a superior end-user experience, but it relies heavily on advanced JavaScript coding

Automatic Completion

To get a better feel for how AJAX can be used in a more realistic scenario, consider a DirectorySearch application. This AJAX example presents the user with a single input text field. When the user begins typing characters into the field, a list of cor-

Listing 4 SQL Query Return

```
package com.ajaxjsf.xml;

// JDBC Based XmlGenerator

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import oracle.xml.sql.query.OracleXMLQuery;

public class XmlGenerator {
  String xmlResponse;

  public XmlGenerator(String input) {

  // Construct query based on input string and return
  // and XML response.

  // The code for this example is Oracle specific
  // and is omitted for brevity.

}
```

The XMLGenerator class called by the AjaxDirectoryService servlet can use any technology, but in this case is coded to use an Oracle database and the Oracle XML SQL feature to return a pure XML string from a SQL query.

Listing 5 The JSF UIComponent

```
package com.ajaxjsf.component;

// Import statements omitted for brevity

public class DirectorySearch extends UIInput
{

  private static final String SCRIPT_RENDERED_FLAG =
    "directory-js-script-rendered";

  public DirectorySearch() {
    setRendererType(null);//This component
      // renders itself
  }

  public void encodeBegin(FacesContext context)
    throws IOException
  {
  String clientId = getClientId(context);
  encodeAjaxJavascript(context);
  encodeInputField(context, clientId);
  }
...
```

The DirectorySearch UIComponent extends the standard UIInput component.

Listing 6 Encoding JavaScript

```
public void encodeAjaxJavascript(
    FacesContext context) throws IOException  {

    String border = (String)getAttributes().get(
      "border");
    String tablebgcolor = (String)getAttributes().get(
      "tablebgcolor");

    // Render Ajax enabled Javascript only once

    if (!jsRenderedFlag(context)) {
      // if not rendered yet, go ahead and do it
      ResponseWriter writer =
        context.getResponseWriter();
      writer.startElement("script", this);
      writer.writeAttribute("type",
        "text/javascript", null);
      render_lookup_function(writer);
      render_processXML_function(writer);
      render_renderTable_function(writer, border,
        tablebgcolor);
      writer.endElement("script");
    }
  }
```

The encodeAjaxJavascript() method includes three JavaScript rendering methods.

responding matches from a fictitious corporation's employee directory appears (see Figure 2). As the user continues typing, the list decreases in size until a best match is found. The user doesn't even have to click on a Submit button as the page updates itself with the most correct matches based on the input so far. This behavior all occurs without requiring a traditional page submission and a complete refresh cycle (see Figure 3). Note that with just a little more JavaScript, the DirectorySearch example could be transformed into the popular autocompletion feature.

The DirectorySearch AJAX example consists of an HTML page, directory.html, and a Java servlet that responds to the AJAX request in XML. The HTML page contains an input text field and a set of JavaScript functions residing in the same HTML page that react to the characters entered, invoke an AJAX request, and update the UI with the response. The first thing to consider in this example is how the AJAX request gets started. It occurs because the input text field in the HTML page has an onkeyup attribute set with a reference to a JavaScript lookup() function:

```
<INPUT TYPE="TEXT" ID=
  "SEARCHFIELD" SIZE="20"
  ONKEYUP="LOOKUP(
  'SEARCHFIELD');">
```

Notice the searchField ID is passed to the lookup() function. It uses this function to determine the current value of the field. The JavaScript lookup() function along with the other functions is embedded inside of a pair of <script> tags in the header of the HTML page. By examining the lookup() function more closely, we see that its purpose is to initiate the AJAX call with a URL of ajaxdirectoryservice, which is the (mapped) name of the AJAX servlet. It uses the input parameter to pass the value of the searchField input field to the AJAX servlet (see Listing 1).

Once the request is initiated in asynchronous mode, a processXMLResponse() function will be invoked once the request is complete. The processXMLResponse() function merely checks to see that the request completed successfully from the servlet and sends it to a rendering function:

```
FUNCTION PROCESSXMLRESPONSE() {
  IF (XMLREQ.READYSTATE == 4) {
    IF (XMLREQ.STATUS == 200) {
    RENDERTABLE();
    }
  }
}
```

The data retrieved from the servlet in this example is pure XML and is based on the input text that was supplied to it. For example, if a string—"sc"—is sent to the servlet, it will respond in XML with:

```
<?XML VERSION = '1.0'?>
<DIRECTORY>
  <EMPLOYEE>
    <NAME>CHRIS SCHULTZ</NAME>
    <TITLE>JSF DEVELOPER</TITLE>
    <EMAIL>CSCHULTZ@FOOCORP.COM
    </EMAIL>
    <PHONE>1-408-555-1234</PHONE>
  </EMPLOYEE>
  <EMPLOYEE>
    <NAME>SCOTT BORLAND</NAME>
    <TITLE>ELECTRICAL ENGINEER
    </TITLE>
    <EMAIL>SBORLAND@FOOCORP.COM
    </EMAIL>
    <PHONE>1-408-555-2468</PHONE>
  </EMPLOYEE>
...
</DIRECTORY>
```

The JavaScript renderTable() function, which is called by the callback function, has the task of iterating through the XML data, constructing an HTML table from it, and appending it to the original HTML page (see Listing 2).

The code may be a bit cryptic, but essentially it just loops through the XML data, constructs a new HTML table with the data in it, and appends it to a specified write location (writeloc), which is an HTML DIV defined just below the input text field. Notice that it clears out the previous HTML results table just before it appends the new one. The DIV that serves as the write location writeloc is defined as:

```
<DIV ID=
  "SEARCHFIELD:TABLE"></DIV>
```

The servlet responds to the request and processes the incoming input parameter (see Listing 3).

Aside from being a typical HTTPServlet, the key things to notice are that the ContentType is set to text/xml and a header parameter, Cache-Control, is set to no-cache. This setting prevents any of the XML data from being cached by the browser. Notice also the XMLGenerator class, which is a custom class that does the XML generation. It can use any technology, but for the example here it's coded to use an Oracle database along with the Oracle

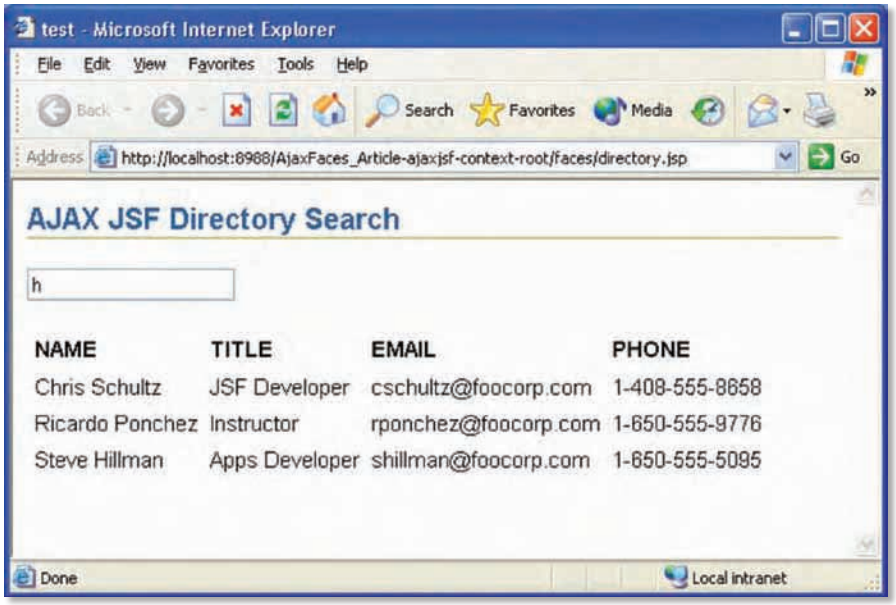


Figure 4 | Déjà Vu An AJAX JSF DirectorySearch component provides a directory lookup just like in the pure AJAX version.

XML SQL feature to return a pure XML string from a SQL query constructed from the input value (see Listing 4).

AJAX-Enabled JSF

What's wrong with the pure AJAX version of DirectorySearch? Having reviewed the architecture of the pure AJAX DirectorySearch application example, we see that the page author bares a very large responsibility for making the AJAX transaction work. As you recall, the AJAX *plumbing* is constructed manually using JavaScript in the HTML client and can easily become very complex. An ideal solution would be to offer a DirectorySearch JSF component where the page author can simply drop it onto a page and it just works.

A JSF version of the DirectorySearch AJAX example is very easy to use. The page author simply adds the directorysearch component to a JSF JSP page:

```
<%@ PAGE CONTENTTYPE="TEXT/HTML"%>
<%@ TAGLIB URI=
  "HTTP://JAVA.SUN.COM/JSF/HTML"
  PREFIX="H"%>
<%@ TAGLIB URI=
  "HTTP://JAVA.SUN.COM/JSF/CORE"
  PREFIX="F"%>
<%@ TAGLIB URI=
  "HTTP://JAVAPRO.COM/AJAXJSF/
  DEMO/COMPONENTS" PREFIX=
  "AJAXJSF" %>
<F:VIEW>
  <HTML>
    <BODY>
      <H:FORM>
        <H2>
          AJAX JSF DIRECTORY
            SEARCH
        </H2>
        <AJAXJSF:DIRECTORYSEARCH
          BORDER="0" />
      </H:FORM>
    </BODY>
  </HTML>
</F:VIEW>
```

At runtime, directorysearch performs its job of providing a directory lookup just like the pure AJAX version (see Figure 4). Since the directorysearch JSF component provides attributes, the user can easily alter certain aspects of the directorysearch without having to edit any JavaScript code. Here's an example of the same component but with a different background. This tag generates the result shown in Figure 5:

```
<AJAXJSF:DIRECTORYSEARCH BORDER=
  "0" TABLEBGCOLOR="#99EEFF" />
```

The architecture of the JSF-enabled DirectorySearch is similar to the pure AJAX example shown previously, except that the JSF component performs the task of rendering the JavaScript directly into the HTML (JSP) page. The JavaScript code then initiates an XML HTTP request to either the same AJAX servlet that was used before, or the AJAX XML response code that was used in the servlet could instead be inserted into the JSF application in two different ways—as a JSF PhaseListener or into the decode() method of the JSF component itself (see Figure 6).

Let's take a look at the key elements of the JSF AJAX version of DirectorySearch. As you can see in Figure 6, the JSF architecture has a JSF component that renders the necessary JavaScript on the client to communicate to the AJAX-enabled service:

```
<AJAXJSF:DIRECTORYSEARCH>
```

The AJAX service, with which the JavaScript communicates, can be implemented in two ways: either it can remain as an independent AJAX servlet, as was used in the pure AJAX example discussed previously, or it can be embedded into the JSF application such as with a PhaseListener or in the component's decode method.

Build It

If the JSF application retains the same AJAX servlet from before, it then has the JSP page, Directory.jsp, and the AJAX servlet, which responds to the XML HTTP requests. The JSP page contains a DirectorySearch tag referencing the underlying UI component and the DirectorySearch UI component that renders both an input field along with the necessary JavaScript to communicate with the AJAX server.

This approach is adapted most easily from the pure AJAX architecture. The only real change is that a custom DirectorySearch JSF component is built, and it takes on the responsibility of rendering the necessary JavaScript at runtime.

Building the DirectorySearch JSF component. For those familiar with JSF custom component development, building a DirectorySearch custom JSF component that renders an input field along with the necessary JavaScript is fairly easy. However, those unfamiliar with custom JSF component development will need some background information on this topic (See Resources

online at www.javapro.com for a link to the article "Building Custom JavaServer Faces UI Components").

In general, custom JSF component development involves creating a UIComponent class that provides the behavior of the component and an optional Renderer class that has code to present or render itself to the client. The Renderer class is optional because the UI component can also contain logic to render itself. For usage in JSP, a custom JSP tag handler (UIComponentTag) must also be created. In the interest of brevity, let's consider only the UIComponent source code, which includes the code that renders the AJAX JavaScript functions. The DirectorySearch UIComponent class has the core structure shown in Listing 5.

Notice that the class extends the standard UIInput component, which is just an input text field, and it also has a setRendererType() method that has a null argument. The null argument just means that a separate renderer is not used with this UI component because it renders itself. Also notice the encodeBegin() method, which is overridden where custom rendering code is placed. The two main tasks of rendering JavaScript and the input field are placed in separate methods. The body of the encodeAjaxJavascript() method is shown in Listing 6.

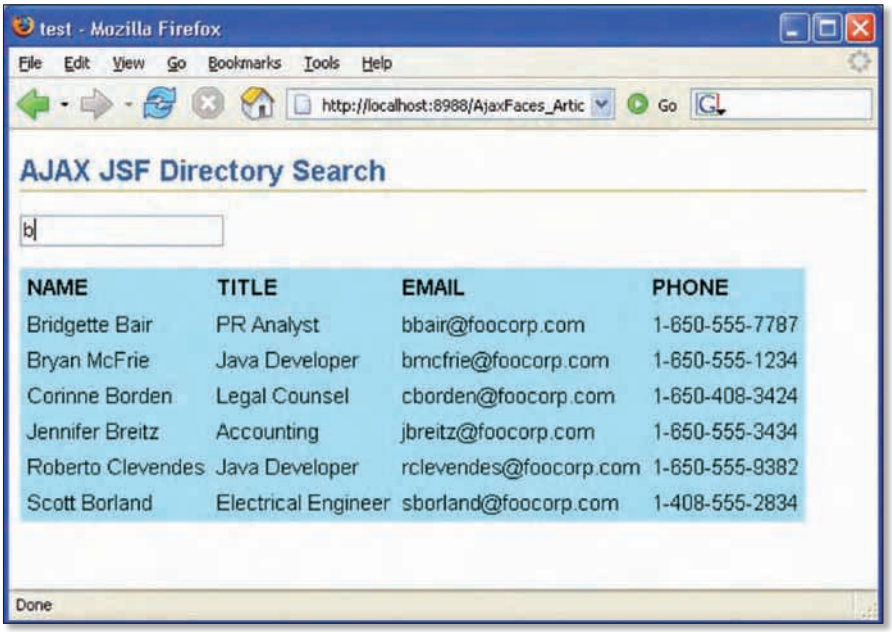


Figure 5 | Feeling Blue The DirectorySearch component with a blue background tag provides this result.

> The AJAX plumbing can be constructed manually using JavaScript in an HTML client but can easily become very complex

Notice in the code the three JavaScript rendering methods: render_lookup_function(), render_processXML_function(), and render_renderTable_function(). Each of these methods renders their respective JavaScript functions shown previously using a ResponseWriter object (writer) that is retrieved from the FacesContext.

Conditionally rendering JavaScript. Before executing the three JavaScript rendering methods, notice that a check is performed first using jsRenderedFlag() to determine whether the AJAX JavaScript functions have already been rendered onto the page. This check is necessary because if this component were to be used more than once on the same page, it would have to make sure it rendered the JavaScript functions only once since other DirectorySearch components on the page could just use the same AJAX JavaScript code. The jsRenderedFlag() method works by placing a marker flag variable onto the Request, and if other instances of this component find this flag they will refrain from rendering an unneeded copy of the JavaScript code. The jsRenderFlag() method returns a boolean value:

```
PUBLIC BOOLEAN  JSRENDEREDFLAG(
  FACESCONTEXT CONTEXT){
    MAP REQUESTMAP = CONTEXT.
      GETEXTERNALCONTEXT().
      GETREQUESTMAP();

    IF (REQUESTMAP.GET(
      SCRIPT_RENDERED_FLAG) ==
      NULL){
      // THIS IS THE FIRST TIME
      // SO RENDER IT
      REQUESTMAP.PUT(
        SCRIPT_RENDERED_FLAG,
        TRUE);
      RETURN FALSE;
    }
    ELSE
      RETURN TRUE;
  }
```

Note that SCRIPT_RENDERED_FLAG is defined earlier in the class as:

```
PRIVATE STATIC FINAL STRING
  SCRIPT_RENDERED_FLAG =
  "DIRECTORY-JS-SCRIPT-RENDERED";
```

Assuming the component hasn't already rendered the AJAX JavaScript functions, the three JavaScript rendering methods will render the functions onto the page using the provided ResponseWriter (writer) object—for example:

```
PUBLIC VOID
  RENDER_LOOKUP_FUNCTION(
  RESPONSEWRITER WRITER) THROWS
  IOEXCEPTION  {
  STRING AJAXURL = "";

  WRITER.WRITE(
    "FUNCTION LOOKUP(FIELD) {\N" +
    "WRITELOC = FIELD +
    \":TABLE\";\N" +
    "VAR SEARCHFIELD =
    DOCUMENT.GETELEMENTBYID(
    FIELD);\N" +
    "\N" +
```

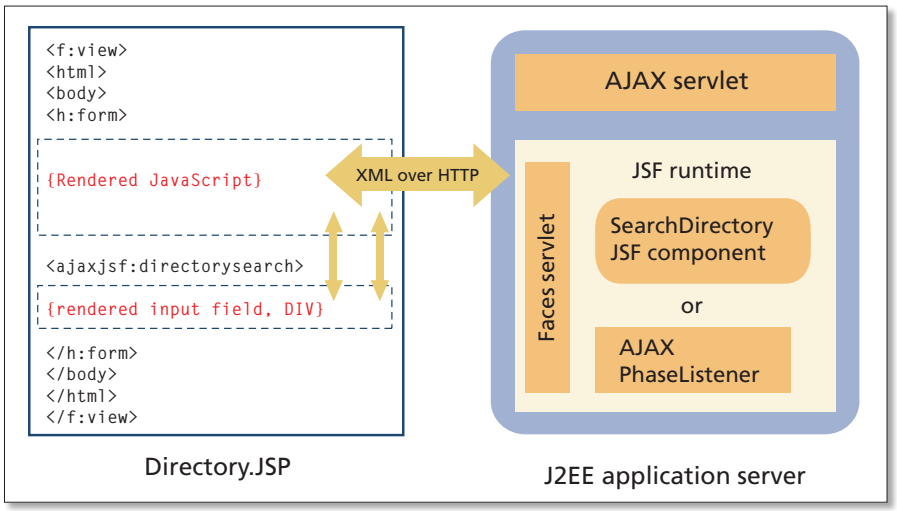


Figure 6 | Pure and Easy The existing AJAX servlet is the easiest architecture if it doesn't need to interact often with the JSF application data. Place the code inside either a JSF PhaseListener or the JSF component itself to have a pure JSF AJAX solution without using a separate servlet.

```
    "IF (DOCUMENT.ALL)\N" +
    "XMLREQ = NEW ACTIVEXOBJECT(
    \"MICROSOFT.XMLHTTP\");\N" +
    "ELSE\N" +
    "XMLREQ = NEW
    XMLHTTPREQUEST();\N" +
    "\N" +
...
```

The source for the remaining JavaScript rendering methods, render_processXML_function() and render_renderTable_function(), are omitted for brevity but are coded in a similar fashion to render_lookup_function().

For a minimal approach to rendering JavaScript, note that an alternative to rendering the JavaScript code onto the page can be achieved by just rendering the JavaScript inclusion string onto the page:

```
<SCRIPT LANGUAGE="JAVASCRIPT" SRC=
  "JS/AJAXSCRIPT.JS"></SCRIPT>
```

When using this approach, make sure that the referred JavaScript file is available to the Web page (such as in a js/ subdirectory). In addition to just placing the JS file in a directory below the HTML root, there are other more sophisticated techniques beyond the scope of this article for dynamically delivering the JavaScript file without the need for copying a JS file under your HTML root directory.

From a conceptual standpoint, that's all you need to understand this version of the JSF-enabled DirectorySearch component because it simply renders the input field and the associated AJAX JavaScript, and the page author doesn't have to bother with it.

Although this approach is the easiest way to implement AJAX with JSF, it still uses a separate AJAX server object. This approach may be adequate for many applications; however, it may be advantageous to integrate the AJAX server object into the JSF application.

> When integrated into the JSF application, the AJAX server logic must listen to the incoming requests, determine if it's an AJAX request, and then respond appropriately

Embedding AJAX Server Logic

There are some possible benefits to integrating the AJAX server logic into the JSF application. These include having direct access to the JSF application data and UI component model, as well as not requiring an additional servlet configuration. When integrated into the JSF application, the AJAX server logic must listen to the incoming requests, determine if it's an AJAX request, and then respond appropriately. Two possible locations to embed this logic are a PhaseListener, which is independent of the component itself and is usable by the entire application, and the component's decode() method, which is the standard location where postback requests are processed. It resides either in the UIComponent or the associated Renderer class.

A JSF application with integrated AJAX server logic has a JSP page, Directory.jsp, or a JSF PhaseListener that is used to process AJAX requests. The JSP page contains a DirectorySearch tag referencing the underlying UI component and the DirectorySearch UI component, which in addition to rendering the component, its decode() method processes the AJAX request.

The specific details on how to implement the AJAX server logic inside of the JSF application are beyond the scope of this discussion, but the general approach remains the same where an incoming request is processed and a response is generated. Details on how to do this implementation can be found in the AJAX-focused chapter (11) in *JavaServer Faces: The Complete Reference* (see Resources online at www.javapro.com); my coauthor of this book, Ed Burns, is also the co-specification lead of the JavaServer Faces specification (JSR 127).

AJAX's hype is valid as it provides the end user with a truly superior client experience; however, implementing AJAX, especially when doing so in a purely manual way, can involve some fairly advanced JavaScript development. Fortunately, the JSF-component model allows for the encapsulation of complex AJAX code into the components themselves, thus greatly simplifying the page author's task of building an AJAX-enabled application. JSF page authors don't need to know how the AJAX JSF components work; they just work! JP

Chris Schalk is a principal product manager and Java evangelist for application server and development tools at Oracle Corporation. Chris maintains a blog on Java EE Web development, and he has also coauthored *JavaServer Faces: The Complete Reference* (McGraw-Hill Osborne Media, 2006) with Ed Burns. Contact Chris through his blog at: **http://jroller.com/page/cschalk**.

Putting Open Source to Work

Properly used, open source code and applications can accelerate the building and deployment of high-quality Java applications

by Peter **VARHOL**

Open source software—in all of its many forms—has been one of the keys to the strategic success of the Java platform and language. The wide availability of high-quality software in source code form has made it possible for developers to leverage the work of others around the world to use development, deployment, and management tools along with code itself to build software more quickly.

In addition to source code that can be incorporated into applications by developers, open source software also provides platforms and middleware that are essential to the building and deployment of Java EE applications. Open source development tools, application servers, servlet engines, and other utilities are used in countless organizations as platforms for applications of all types, including those critical for business.

However, using open source opens up a host of considerations for developers building commercial or internal applications. Software quality, bug fixing, and support are all issues that are different when employing open source, and in some cases the validity of the open source approach to software has been called into question. Those issues haven't stopped development teams from deploying applications using open source, or applications deployed on open source platforms, and use continues to grow rapidly.

There is much more to open source software than using it to build and run applications. It's also a movement that defines how software is created and distributed among developers and end users. In the aggregate, open source represents an alternative business model for software development, one that is seemingly at odds with the capitalist roots of its primary commercial purveyors. As a result, commercial software vendors treat open source warily, reluctant to embrace it but unable to ignore it.

Many questions remain about open source. Those questions haven't prevented millions of developers and hundreds of software vendors from adopting open source, either as a strategy or simply because it was easier to get things done with open source. Whether use was planned or unplanned, it has become a force in the Java community that can't be ignored.

Why Open Source?

Software developers have always shared the product of their work. Part of that sharing is professional pride; there are few better venues for technical problem solving than writing software. And those who are successful at it can use open source as a way to promote their professional standing and enhance their employment prospects. There are few better ways of demonstrating competence than writing code that your peers choose to use.

And software developers simply like solving problems. Providing their solutions as open source is probably the best means for those solutions to be used by others. In many cases, a solution provided by open source isn't a product in the traditional sense, but an implementation of an algorithm or a specific way of accomplishing a particular common activity.

Open source software, in any form, however, is disruptive. Organizations using it are wary of its legal standing (see the sidebar, "The Legal Side"). Developers writing it take time and energy that

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0601 Download all the code for this issue.

Read More

JP0601PV_T Read this article online.

JP0501DH_T Read the related article "Pushing Portal Potential" by David Hritz.

JP040818LT_T Read the related article "3 Tips for Developing on Linux" by Adam Kolawa and Jeehong Min.

JP040609WK_T Read the related article "Make Mine Large" by William Knight.

could be devoted to paying work, or to other parts of their lives, and software vendors see both direct (and perhaps unfair) competition and an uncertainty of how to leverage open source for their own uses.

Another reason there is a mystique to open source is that even today no one knows for sure how to price the product in question. Despite the very real expertise and ability it takes to create software, it is an intellectual product rather than a physical one. Because pricing is so uncertain, it tends to be all over the map, from almost nothing to extremely expensive, and it was probably inevitable that someone would price it at nothing.

That is not quite true, of course. Open source software can be sold. The typical definition for open source is that whatever the price is, that price includes the source code. In practice, many give it away and make their money in other ways.

The theory and economic model of open source originated with Richard Stallman, MacArthur Foundation Fellowship recipient and founder of the Free Software Foundation. He defined this model in a document called The GNU Manifesto (see Resources online at www.javapro.com). Even today most would consider it almost quaintly naïve—a description of the world as someone would like it to be rather than the way it was.

For a decade or more, Stallman and the Free Software Foundation were largely ignored by the software industry. Stallman and his colleagues are talented software developers in their own right, and they wrote a number of quite good albeit difficult-to-use development tools bearing letter-names such as GCC, GDB, and EMACS.

Then several events happened to change the fate of this obscure group. First was the launch of Linux, the operating system kernel written by Linus Torvalds as a computer science student. Many people have written operating systems; only Torvalds licensed his under Stallman's GNU General Public License (GPL). The GPL provides for the free and unfettered distribution of software as source code, for the first time providing for rules that govern how open source is obtained and used.

The second event that happened was the opening up of the Internet for public use. An amalgamation of several government-funded networks, the Internet had only been available for use by academics (including students) and government researchers. Legislation made the Internet a publicly-available resource, and provided a much-needed distribution mechanism for both Linux and other open source software.

The last event was the launch of the Java language and platform. A product of its times, Java seemed built specifically for Internet applications. And as an alternative to the seemingly growing hegemony of Microsoft, both software vendors and individuals rushed to use it to build applications, applets, and tools. As a result, Java code proliferated quickly around the world upon its introduction.

The result was that Java code or applications that work with Java now make up a large amount of open source commonly available. Much of that software has been made available through organized sources, such as the Free Software Foundation, Apache Foundation, or SourceForge, but it is still possible to find individual applications or code snippets on thousands of different Web sites (see Resources online at www.javapro.com).

Business Models Proliferate

Many developers are driven by goals other than money to write open source code. No developer would deny that there is a certain status in writing a successful open source application or utility. However, some individuals and many software companies employ open source as part of a business strategy.

Most traditional business models are straightforward: a company buys raw material, processes that material, and sells the end product for a profit. Software in general is a little more abstract, in that the raw material is an idea and the end product is intellectual property. Open source, however, is still more abstract, in that a company is typically giving away both the end product and the raw material for making it. Only the pro-

The Legal Side

Open Source software isn't a one-way street. At the very least, you have the licenses to contend with—possibly many licenses. According to the Open Source Initiative (OSI), there are over 50 different licenses that qualify as open source (see Resources online at www.javapro.com). These licenses govern how software that is licensed under open source can be obtained, used, and distributed. They are alike in that they share similar goals, but almost invariably they have subtle differences in their requirements for handling code.

Many organizations use open source on internal development efforts with no intent to redistribute those applications, and they often don't bother to consider the license. However, code often has a way of slipping into other applications and into commercial products or services, and the organization may not know it until the license has clearly been compromised.

All of these legal issues mean that anyone who consumes open source must be cognizant that they are doing so and understand the requirements of the license or licenses with which they are made available. Consider forming a committee consisting of developers, architects, line-of-business owners, and legal professionals to assess each use of open source, both in an application and as a part of the infrastructure. By being proactive, a development organization can prevent pain later down the road. Specifically:

- Read and understand the open source license.
- Based on the license, establish procedures for using the application or code.
- Monitor those procedures over the life of the project.

Consulting an attorney is useful, but doing so should not necessarily be the basis of a decision. A lawyer will advise on legal risk, not on whether the cost or utility of the software outweighs any possible risk. The technical people are the ones who need to understand the risks and benefits, and make the right decision.

cess remains intact, perhaps because it is the effort of many individual developers that could not likely be duplicated in any case.

Thus it is no surprise that companies are still experimenting with business models that utilize open source software. Among the first was Red Hat, which consisted of packaging and reselling a Linux distribution at a nominal price, and then charging for support. Red Hat has since expanded that model to add features to Linux required by enterprise computing environments.

A variation on that model is the one being promoted by JBoss, termed "professional open source." JBoss pays its development teams to create open source software, which is available for free download, or through a paid subscription. In addition, JBoss offers consulting, training, certification, and migration services. MyEclipse is yet another different take on the same approach, offering testing and support for a specific configuration of the Eclipse IDE. These approaches can be profitable, although it is not clear at this point if the models can scale to large companies with multiple projects across different lines of business.

Offering different licenses for different purposes is another business strategy. One example is MySQL, which offers a GPL version for use with other open source software, and a commercial version for use with commercial or proprietary software.

On the other side of the equation, users face their own uncertainties. Oddly enough, cost is near the top of the list. For consumers of open source, this seemingly free software resource can come with hidden costs. You may find yourself on your own with an open source application, responsible for installing, servicing, patching, updating, and troubleshooting it. While you can engage a vendor to do these activities for some of the more popular open source applications, if you select a less popular one, or just use code snippets, you may have to develop substantial in-house expertise.

It is an ongoing debate as to whether adopting open source for production applications is more or less costly than commercial equivalents. Intuitively, open source seems like it should be less expensive, but total cost of ownership calculations take into account a wide variety of factors, such as the availability of training, consulting, and support services.

Another potential danger of open source software for users is the question of intellectual property rights. Many commercial software firms have taken on a strategy of patenting software innovations to build up a library of protected intellectual property. In many cases this tactic is for self-protection, in case that company is faced with claims against it by others.

However, there is the growing fear that these patent libraries can be used against open source efforts. Open source innovations are rarely announced or patented, and those holding patents covering those innovations can use them as legal tools. Competitive commercial vendors may well sue open source developers over specific implementations, targeting a population that lacks the resources to fight back in the court system. And user organizations, which can also face lawsuits, are typically risk-adverse, and prefer to avoid lengthy court battles on their technology choices.

> There is much more to open source software than using it to build and run applications

While this danger has little to do with the technical merits of open source, it represents a true threat to the open source model. Loss of legal use of major open source software seems unlikely at the moment; intellectual property rights represent perhaps the biggest deterrent to use today.

Building a Java Platform

Using open source code in your own applications is one way developers are tapping this resource. However, using it as infrastructure to build and run applications is probably the more common use, at least for major open source projects.

Java is a specific and primary beneficiary of this use. Unlike most other languages, it is more than possible to build a full development-and-deployment platform for Java entirely from open source software. In fact, it is more than possible; in many cases it represents best-of-breed choices.

For development, both Eclipse and NetBeans are fine alternatives. Eclipse has the larger community and more add-ins, but the most recent versions of NetBeans have gotten good reviews, and both appear to be quality environments for building Java EE applications. For those seeking a level of testing and support for specific configurations, MyEclipse provides a standard Eclipse platform with a specific configuration of open source tools for a nominal price. MyEclipse incorporates Struts, JavaDoc, JavaServer Pages (JSP) design, JavaServer Faces (JSF), Hibernate object-relational persistence, and a host of other quality open source features. Yes, they are all freely available, but MyEclipse does a nice integration and installation, along with support.

Many other development and process tools are available through open source, either as Eclipse add-ins or as stand-alone tools. Repositories and management facilities for requirements and source code control systems can round out a professional-level development process built on open source solutions.

The infrastructure and middleware areas are fertile ground for IT groups seeking development and deployment platforms. Tomcat is used extensively by developers as a servlet engine, and is even occasionally used in deployments. For more production-oriented application servers, JBoss is comparable in many ways to commercial counterparts, while Apache sees predominant use as a Web server. While Apache is not specific to Java, it is often used as a front end for Web-based Java applications, and the technologies used with it are also compatible with the Java platform.

The plethora of open source platform tools has to beg the question of why any organization would consider paying for commercial products that provide similar capabilities. With companies such as Red Hat and MyEclipse providing testing and support services for specific configurations, is there still a role for the traditional software tools or platform vendor?

Well, yes. In addition to the ongoing debates surrounding total cost of owner-

ship, it is likely that innovation and feature decisions are made differently between the two models. And there is no question that some users prefer the level of integration found in the offerings of a specific vendor, or simply prefer having one vendor with which to interact. From a feature standpoint, however, open source solutions tend to stack up very well.

Does Open Source Make Sense?

A variety of constituents contend that open source development and use has a number of negative characteristics. Some claim that the process results in software that is no better than, and often inferior to, traditional development models. Others go so far as to claim that open source almost by definition is anticapitalist and a menace to the free market (by which they typically mean for-profit software vendors).

Such claims seem excessive, but what is clear is that open source represents a real alternative to commercially developed software. The legal and intellectual property uncertainties do not seem to have made a significant dent in adoption. And Eric Raymond, open source developer and evangelist, among others, has made compelling arguments that open source has the potential to be at least as high quality as its commercial brethren.

Yet anyone using or considering using open source software should do so with their eyes open. There are legal risks that will not likely be resolved for years, if at all. And while the use of open source software can help prevent the feared "vendor lock in," it may simply substitute one type of dependency for another.

There are also legal and ethical responsibilities related to using open source software. If you are a consumer of open source, the best way to think about it is as a transaction. Specifically, you don't get something for free, despite the fact that open source is often free. However, unlike commercial transactions, you are probably not paying with money, and you may not be paying the developer or provider, at least not directly.

Most open source licenses require the source code to be propagated with any redistribution. There is a certain sense of security in having the raw material of your software, so to speak, but that may be false security. Having the source code doesn't necessarily mean that you can pick it up and branch it away from the open source mainstream. If that code becomes part of a commercial product, it may have to be released as open source. In any case, it may make both business and ethical sense to contribute back to the open source community.

Any organization using open source software, or considering its use, should have a committee to evaluate the selection and use of software on a case-by-case basis, and to determine any contributions to open source. With an organized approach to evaluating and using open source code and applications, both the using organization and the community will benefit. JP

Peter Varhol is a senior member of the technical staff for Progress Software. Contact Peter at **peterv@mv.mv.com**.

JSTL Gives Web Applications Flexibility

While scripting languages like PHP continue to draw wide appeal, there are downsides. Try JSP tag libraries for a faster approach

by Alan **BERG**

Scripting languages are quick to develop and easy to manipulate. If you want to build a dynamic Web site in a day or make quick changes to CGI-like scripts, then with relatively few learning hours you can apply scripting languages to these tasks. Further, with scripting languages like PHP, a server-side scripting language that is vaguely similar to C, you gain the potential for rapid development cycles. Once you enact a change, the change propagates live.

These are excellent characteristics of PHP, and what's even better is the language has the reputation of running fast. However, on the dark side there are some drawbacks. Without rigid control, code grows unruly and may become costly to maintain and hard to debug. The diffusion of logic and HTML presentation tags can make for pronounced readability issues. The same issue is true for JSP coding. Struts and other high-level Java frameworks that use the Model-View-Controller (MVC) pattern combat the maintenance issues by dividing applications in such a way that Web designers are not bothered with Java coding. Further, Java coders avoid the majority of the interaction design and graphical user interface (GUI) coding.

However, this separation of tasks between the GUI designer and programmer also has a price; the frameworks have a significant learning curve and indirectly expect a team-like infrastructure that is commonly only found in mid-size to large projects. For small companies the learning curve associated with the frameworks sometimes represents an unrealistic barrier. We may need to be better at addressing the needs of the small businesses and small office/home offices (SOHOs). One possibility for rapid application cycles and instant code changes is pure JavaServer Page (JSP) coding, including all the potential readability and growth pains the practice may entail.

Easier Web Apps

The JavaServer Standard Tag Library (JSTL) is a superset of JSP that attempts to encapsulate commonly required functionality within a series of four extra tag libraries. The idea behind it is to allow for easier creation of Web applications without resorting to mixing Java code in with tags so quickly. Common Web application functionality is encapsulated. JSP has an expression language (EL) that allows for easier plumbing to sessions and beans. For example, the expression:

```
${sessionScope.driver}
```

obtains the value of the attribute driver from the scope session. Without EL some of the basic JSP coding is verbose. For example, obtaining a property from a bean in EL is similar to ${login.UID}, where as without EL the resulting code is:

```
<%= ((Login) pageContext.
  getAttribute("login")).getUID
  () %>
```

Note that EL is supported by JSTL but can function outside the extra tag libraries as well. The four logically divided JSTL tag libraries are: core, which works with basic programming constructs such as

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0601 Download all the code for this issue.

JP0601AB Download the code for this article separately. This code demonstrates applying JSTL to make connections in a database.

Read More

JP0601AB_T Read this article online.

JP030109BK_T Read the related article "Develop Faster with Tag Libraries" by Budi Kurniawan.

EA0302SG_T Read the related article "View from the Summit" by Steve Gillmor.

JP0103KJ_T Read the related article "Put Up a Good Front" by Kevin Jones.

Listing 1 Simple PHP

```
<?php
include 'config.php';
include 'connection.php';
$query = 'select * from parts';
$result = mysql_query($query) or die(
  'Sorry the query failed');
while($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
  $temp= "uid :{$row['uid']} <br>Name : {$row['name']}
    <br>" .
      "Description : {$row['description']} <br><br>";
  echo $temp;
  $message=$message.$temp;

}
$message=$top.$message.$bottom;
 mail("$to", "$subject", "$message", "$headers");
include 'close.php';
?>
```

This simplified PHP application shows discrete areas of functionality that are separated over a number of pages.

Listing 2 Define the Tag Library

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD JSP Tag Library
  1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>1.2</jsp-version>
<short-name>Article</short-name>
<uri>article.tld</uri>
<display-name>Simple Tag Example</display-name>
<description>A tag hiding some coding from a web page
  designer</description>
<tag>
  <name>debug</name>
  <tagclass>playground.berg.tag.Debug</tagclass>
  <info>Print some debug information</info>
  <bodycontent>empty</bodycontent>
</tag>
</taglib>
```

The Tag Library Definition for the debug tag is shown in this XML snippet.

iterations; internationalization and formatting, which allows for the loading and use of resource bundles for internationalization and the general manipulation of string formatting; SQL, which enables connection to databases through data sources as well as the general manipulation of databases; and XML, which is used for generic manipulations. Note here that with up-and-coming technologies such as AJAX that make extensive use of XML, expect to see more use of the XML library.

> The idea behind JSTL is to allow for easier creation of Web applications without resorting to mixing Java code in with tags so quickly

Take a look at a simplified PHP page (see Listing 1). Specific functionality is separated into pages. The config.php file contains global variables, and connection.php makes a connection to a specific MySQL server and then closes the connection.

As suggested by the name, $query holds the SQL query. If you have queries that may change over time, then it is better to have each query stored in a text file and have the query read in by the fopen() method. Doing so allows your local database guru the opportunity to manipulate queries that are not entangled in the presentation.

The line starting with $result performs the query by calling the mysq_query() method, which is buffered, and rendering of the HTML needs to wait until all the results are returned. An unbuffered version of this method also exists. If the method fails to obtain a connection the text "Sorry the query failed" displays. It would be nicer to redirect the user to a standard error page at the point of failure instead of inconsistent print statements. PHP is a programming language that supports iteration; the while statement iterates through an array of results collected by the Mysql_fetch_array() method. The array indexes in $row are the column names of the Mysql table, and $message holds a string version of the results in HTML format. Mail is the standard method for sending mail within PHP. The preconfigured variables are hidden in the config.php file. The header shown here states that the message sent will be in HTML format:

```
$headers =
  "From: $from\nReply-To:
  $replyto\nContent-Type:
  text/html;
  charset=iso-8859-1";
```

Consider some salient issues here: The Mysql functionality is not enabled by default and needs to be enabled through the php.ini file. The functionality is then globally enabled. Secondly, many PHP Web sites do not use connection pooling, and therefore there is a one-on-one relationship with connections and page load.

Roll Your Own JSP Tag

Rolling your own JSP tag is a straightforward method for encapsulating Java code in JSP that gives you the advantage of reuse and ease of coding. The disadvantage is programming effort and maintenance costs. Imagine the commonly found Java issue of class path conflicts. On one server, your SQL statements are being fully understood. However, on another server an error

Listing 3 HTML System Properties

```
package playground.berg.tag;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.IOException;
import java.util.*;

public class Debug extends BodyTagSupport {

  public int doStartTag() throws JspException {
    JspWriter out = pageContext.getOut();
    try {
      out.println(systemPropertiesAsHTMLTable());
    } catch (IOException e) {
      throw new JspTagException(
        "Debug tag error: "+e.getClass().getName());
    }
    return SKIP_BODY;
  }

  private String systemPropertiesAsHTMLTable(){
    String message=
      "<h3>Checking System Properties.....</h3>";
    message+="<table border=1 width=\"80%\"    align=
      \"center\"><tr><td><b>Name</b></td><td>Value
      </td></tr>\n";
    Properties properties =System.getProperties();
      Enumeration enumeration =
        properties.propertyNames();
    for (; enumeration.hasMoreElements(); ) {
      String name = (String)enumeration.nextElement();
      String value = (String)properties.get(name);
      message+="\t<tr><td><b><i>"+name+"</i></b>
        </td><td><i>"+value+"</i><td></tr>\n";
  }
      message+="</table>";
    return message;
  }
}
```

Here is the java code called within the info tag.

is thrown that complains about Unicode in the returned query results. In the back of the collective head is the idea that perhaps the database driver versions have somehow misaligned. It would be nice in such a situation to quickly produce a debug screen through the code shown in Listings 2 and 3. To create a functional tag library you will need to follow a couple of steps:

1. *Write a tag library definition.* The definition explains to the servlet container the allowed syntax of the tag library (see Listing 2). The general properties of the TLD are defined, and then the specific tag. The tag's name is defined by <tag><name>, and the java class is called by <tag><tagclass>.
2. *Write the personalized java class.* The class mentioned in the TLD can take advantage of the javax.servlet.jsp.tagext.BodyTagSupport class and extend overriding the doStartTag() method to generate output when the tag is called from within the JSP page. Note that different methods are declared at different points in the processing of the information contained in an instance of the tag. However, because we have stated in the TLD that the tag has no content by <bodycontent>empty</bodycontent>, we need to override only one method.

The systemPropertiesAsHTMLTable() method iterates through all the system properties and generates an HTML table stored in a string with the name and value pair of the property. Notice the bad practice of having formatting hard coded inside the method. It is significantly better to mention a style sheet and pass the location of the style sheet as a parameter in the tag. However, the extra complexity doesn't fit within the current story.

It's worth mentioning where to find the TLD in the WEB-INF/web.xml file. If you have placed the TLD in the WEB-INF directory and then in web.xml, you need to add:

```
<taglib>
  <taglib-uri>article.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/article.tld
    </taglib-location>
</taglib>
```

Notice that within the TLD the <taglib-uri> is also mentioned through the <uri> tag. Next, add the tag library to the JSP and call it:

```
<%@ taglib uri=
  "/WEB-INF/article.tld" prefix=
  "info" %>
<info:debug/>
```

We can conclude that although it is relatively straightforward to write your own JSP tag libraries for simple functionality, the complexity also increases when moving to slightly more difficult situations. Why bother when JSTL can do most of the heavy lifting?

Time Out

Before moving on to the coding, let's take a moment to discuss bad error handling. How many times have you seen a Server 500 error when you trawl the Internet? Worse still, the full stack trace of the exception on the page is totally alien and meaningless to the average visitor. Or in the case of PHP, an error message with the line number and method call is also not nice, but at least it doesn't fill the page with gibberish. Things happen, Murphy is alive and very active. Infrastructure fails. Sites get too busy. Memory is consumed. Database connections fail.

If you code for run-time errors you may make the code base unwieldy and full of try catch blocks and print statements. Under Tomcat, the tag example we developed previously fails if a security manager is in place because only a limited number of properties are allowed from the standard conf/catalina.policy file. A compromise is to catch all exceptions in the JSP, and redirect them to a second page. In the page you can log all errors or even send mail and so on and then also print out a more intelligible message, which is achieved through setting:

```
<%@ page errorPage="error.jsp" %>
```

at the top of each displayed page and writing an error.jsp page similar to this:

THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

Come visit the Journal's new home at **www.ArchitectureJournal.net** live this December. The new site contains a full library of articles from previous Journal issues in addition to upcoming highlights of our next issue. Browse the content today and post comments and letters directly to the editor!

Now live at www.ArchitectureJournal.net!

Microsoft®

ARC

Listing 4 HTML-Mailing JSP

```
1 <%@ page errorPage="error.jsp" %>
2 <%@ taglib uri=
  "http://jakarta.apache.org/taglibs/mailer-1.1"
  prefix="mt" %>
3 <%@ taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c" %>
4 <%@ taglib uri="http://java.sun.com/jsp/jstl/sql"
  prefix="sql" %>
5 <%@ include file="config.jsp" %>
6
7 <h3>Success....</h3>
8
9 <sql:setDataSource
10  driver="${sessionScope.driver}"
11  url="${sessionScope.url}"
12  user="${sessionScope.user}"  password=
  "${sessionScope.password}"/>
13
14  <sql:query var='answer'  >
15    select * from parts
16  </sql:query>
17
18  <mt:mail>
19    <mt:subject>JSP Example</mt:subject>
20    <mt:setrecipient type="to">
        ${sessionScope.mailAddress}</mt:setrecipient>
21    <mt:from>${sessionScope.mailAddress}</mt:from>
22    <mt:server>${sessionScope.mailServer}<
        /mt:server>
23    <mt:message type="html" >
        <h3>Order Information</h3>
24    <c:forEach var='row' items=
        '${answer.rowsByIndex}'>
25      <c:forEach var='column' items='${row}'>
26        <c:out value='${column}'/><br>
27      </c:forEach>
28    <br>
29    </c:forEach>
30    </mt:message>
31  <mt:send>
32    The following error has occured:
33    <mt:error id="err">
34    <jsp:getProperty name="err" property="error"/>
35    </mt:error>
36  </mt:send>
37  </mt:mail>
```

This JSP reads information from a database and then sends a formatted HTML document by e-mail.

```
<%@ page isErrorPage="true" %>
<HTML>
<HEAD><TITLE>Error Page
  </TITLE></HEAD>
<BODY>
<H3>Exception Information</H3>
<%= exception %>
<%
Error handling code.
%>
</BODY>
</HTML>
```

If you feel like challenging yourself, you could write an error-handling tag library that prints pretty exception pages. You can download an extra tag that simply throws an exception (see Resources online at www.javapro.com). If you wish to see how a particular Web application looks on failure to the average user, install the tag library and place the tag on the pages you our interested in, for example:

```
<%@ page errorPage="error.jsp" %>
<%@ taglib uri=
  "/WEB-INF/article.tld" prefix=
  "info" %>
<info:ExpressError/>
```

If the same PHP functionality you see in Listing 1 is enacted in JSP alone, then we would need to be able to get configuration information into a JSP, get information out of databases, iterate through the results, and then send an HTML document through SMTP. Further, it would be nice to have an obvious path to migrating to heavier frameworks later as the Web site grows in complexity and popularity.

The Jakarata taglib project is one potential solution. This project is an Apache derivative. The project contains a diverse list of helper tag libraries, which are constantly maintained and improved by a dedicated following of like-minded developers. The taglib project is expanding over time, and it is well worth the effort to visit the project Web site regularly. The two sets of libraries that will be mentioned here are a JSTL implementation and a special mailer tag set that allows for easy e-mailing.

Note that the TLDs are stored in the downloaded JAR files and do not have to be mentioned in web.xml.

A Piece of the Configuration

The first part of the puzzle to solve is how to pass configuration information. As we see in the PHP application, global constants may be kept in a separate page. This functional separation has the advantage (for PHP) of keeping the code and configuration separate. Of course, you can keep configuration in many places including property files, web.xml files, database files, and so on, but for the sake of comparison we will use a similar methodology. This code snippet uses the core library to set up an attribute named url that has session scope:

```
<%@ page errorPage="error.jsp" %>
<%@ taglib prefix="c" uri=
  "http://java.sun.com/jsp/jstl/
  core" %>
<c:set var="url" value=
  "jdbc:mysql://localhost:3306/
  orders" scope="session"/>
```

By including the relevant file that contains this code snippet, the first page the user visits through:

```
<%@ include file="config.jsp" %>
```

guarantees that the session is fully populated. Line 5 in the core application (see Listing 4) loads in configuration information. Lines 9–14, starting with <sql:setDataSource, use the configuration information to define a datasource for connecting to the database. In the case of the included project the datasource is a single JDBC connection. However, you may link in JNDI resources through this tag as well. One issue with the tag is that at the top of the web.xml file you need to have JSP version 2.4 defined, and absolutely no version lower or the current core taglibrary will fail to function correctly.

The <sql:query lines (14–16) perform a query using the default datasource. The resultset is stored in the attribute answer ready for manipulation. The <mt:mail> lines (18–37) are quite long because the message is being generated by iterating through the resultset, referenced in the answer attribute. The global properties are defined such as the host and from address. The message type may be either text or HTML.

Within the <mt:message> tag (lines 23–30) the database results are iterated through by the core tag library, and the results are written through a combination of plain HTML and c:out. If an error occurs, then lines 33–35 will print an error message.

We've put together well-known tag libraries to mimic similar work that was enacted by a simple PHP application. The most complex part of the page is the iterating through the database resultset. The method of generating the message string is a mater of taste. For example, you could include Java to populate a string, and place the result through <c:out in the message. This extra coding complexity is part of the JSTL learning curve.

Being agile and able to achieve rapid application cycles and modifications of standard Web sites on the fly requires rapid build cycles. JSP, Active Server Pages (ASP), and PHP are competing head to head. By working with the JSTL and other well-known tag libraries, rapid application development (RAD) is supported. Important side issues in the competition include the quality of the integrated development environment (IDE) and the simplicity of setting up the application servers.

The popular Eclipse environment doesn't provide out-of-the-box support for JSP, but this situation may change soon. Currently there are commercial and noncommercial Eclipse plug-ins. MyEclipse or Nitrox, for example, support JSP editing. Linux, Apache, MySQL, and PHP (LAMP) servers are relatively easy to set up and understand. Infrastructure based on JBoss or Tomcat has that potential for simplicity. However, IIS6 servers win on ease of installation. If Java is to be king in this field, then that is where the focus may need to turn.

Exit This Way

Arguably, where Java wins is in terms of escape routes. Java is mature and built for use in the enterprise. Many frameworks exist that are helpful for scaling up. Success does happen. Teams get bigger; effort starts to get divided. Then the full story of dividing responsibility comes to the fore. Frameworks such as JavaServer Faces (JSF) and Struts can be migrated too without extreme fuss. The use of managed beans in JSF, where you can define bean creation and scope within XML files, can take away some of the drudgery and repetition of code. You may even get to play with their cool tag libraries. For future reference, yet another Apache project is Myfaces, which is an implementation of JSF. The project includes some nice tags such as master view tables and is even expanding into the AJAX arena.

We've taken a whirlwind tour of the possibilities for making a basic Web application rapidly. JSTL in combination with other tag libraries is well suited for this task. It can be argued that iteration in the core tag library requires a small learning curve. Further, if later the functionality is expanded and larger team groupings occur, there are escape routes to more monolithic frameworks such as JSF, spring, and Struts. *JP*

Alan Berg, Bsc., MSc,.PGCE, has been a lead developer at the Central Computer Services at the University of Amsterdam for the last seven years. In his spare time he writes computer articles. He has a degree, two masters, and a teaching qualification. In previous incarnations he was a technical writer, an Internet/Linux course writer, and a science teacher. Contact Alan at **reply.to.berg@chello.nl**.

Fast or Good?

Write code fast or write it to last. Your choice boils down to the trade-offs and compromises inherent in computing

by Peter **VARHOL**

A couple of things I've read recently have made me look once again at the whole problem of advances in software development and productivity. The first was the serialization of a talk given by Charles Petzold to the New York City .Net User's Group titled, "Does Visual Studio Rot the Mind?" The second was a blog post written by Joel Spolsky titled, "The Perils of JavaSchools" (see Resources online at www.javapro.com for links to both sites).

Petzold might be considered to be the dean of Windows programming authors, while Spolsky's Joel on Software site is one of the most coherent and widely-read blogs on software development and the software business around. That both focus on development using Microsoft languages and tools is mostly irrelevant, and Petzhold's comments on Visual Studio could be applied to recent advances for any IDE.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download
JP0601 Download all the code for this issue.

Comment
www.fawcette.com/weblogger/forum.aspx?id=11
Read and respond to Peter's blog.

Read More
JP06010E_T Read this article online.

JP0411OE_T Read the related article "Put Debugging to the Test" by Peter Varhol.

JP0406PV_T Read the related article "Building a Better Application Life Cycle" by Peter Varhol.

FOSRLM_T Read the related article "Design for Success" by Edmund X. DeJesus.

Let's start with Spolsky. He notes that a number of universities offering computer science degrees have based their curricula more or less exclusively on Java, which might seem to be a good thing to the readers of these pages. However, Spolsky notes that Java is simply too easy a language on which to base a foundation for learning computer science. He may have a point, in that programming in Java is not a good way of learning about how executing code interacts with memory, for example, since the JVM largely abstracts memory utilization from the coder.

As an employer (he is founder and head of Fog Creek Software), Spolsky also claims that Java is simply too easy to enable him to distinguish between average programmers and great ones. I don't think that he has anything against managed languages in general, but strongly makes the point that any computer science education without working with pointers and lambda calculus is inadequate. A fundamental understanding of how computers really work may not be possible with Java.

Going with No Flow

Petzold describes how Visual Studio tools make it possible to write fast code, but not necessarily good code. Visual Studio (and to be fair, other IDEs, including Java IDEs) treat applications not as one continuous flow of logic, but rather as a set of code snippets associated with user-interface elements. There can be no sense of the flow of execution because too much of the code has been generated rather than written.

Likewise, generated code can be problematic. I don't think he's necessarily against it (although he does seem to prefer writing his own), but he does note that there is no chance to inspect or change it. Visual Studio also makes assumptions about what files and libraries need to be included in a project, and it often adds things that are simply not needed. In addition to code bloat, this characteristic makes it hard to maintain the application over time, as future programmers assume that files were added for specific reasons.

I often wrestle between the need to be productive and the need to understand what is happening in an application in a deep sense. The business developer within me knows that applications have to be completed and put into productive use more quickly than ever. And as those applications get more complex, the tools have to provide a higher level of productivity aids to meet rapidly changing business needs.

From this perspective, the modern IDE is great. We don't spend a lot of time writing code that the computer or operating system, but not the application, requires. The "software backlog" of up to three years that was the bane of the 1980s and 1990s has largely vanished, thanks to the improved productivity of developers using modern tools. While it still takes time to write software, there are many possible ways of doing it so that tight schedules can be met.

Modeling and other emerging techniques that enable us to define our solutions in closer context to the business problem are essential to the very practical demand of meeting the business requirement. All of these are good things that help make businesses more responsive to changing market conditions and customer needs.

However, the educator in me demands understanding rather than rote learning. Working at a higher level of abstraction

afforded by the modern IDE works most of the time, but a lack of understanding of the full execution model and underlying mechanisms can at best make for some poor design and implementation decisions. At worst, it can eliminate any chance of getting the application correct at all. Most abstractions in computer science are leaky, and expose the low-level details at inopportune times, making an understanding of those details pretty important.

Career or Job

The same virtual machine that powers Java and makes cross-platform development possible also hides the vague nature of memory, making it easier to write applications and find bugs. That efficiency is usually a good thing, except when it's not. Because memory management is hidden, it is possible to write code that uses memory inefficiently. For example, programmers may not be aware that unlike unmanaged languages like C, allocation is cheap and deallocation is expensive. A C programmer may be used to allocating large blocks of memory at once, which can be a reasonable performance strategy in that language, but in Java, you pay for that strategy when it comes time to garbage collect that memory.

These are areas of learning that should remain relatively constant over time, acting as a foundation for a career rather than as training for a job. Subject areas like memory organization, formal language, data structures, and systems programming are essential to understanding computers and software in general, irrespective of the hardware architecture or programming language. For example, I did my initial systems programming course on a Control Data Corporation Cyber, yet it enabled me to understand and work with a number of different processor architectures over the years.

A good education should be able to last an adult lifetime. A good programming language, on the other hand, is at best serviceable for a decade. Increasingly, programmers need to know more than one language, especially if they work on Web applications. To specialize only in a single language or technology may be the quickest route to obsolescence in a career.

In reality, these two points of view between fast code and good code are discussing two different issues. At one extreme are concerns about the minimal requirements to get the job done in the fastest possible time, while at the other are concerns about the qualities of the software and the robustness of the process that produced it. For fast development, you use tools and techniques designed for speed. Often, these are abstracted away from the need to understand the execution model of the application, so that the focus can be on quickly solving the business problem.

Of course, it doesn't always work out as ideal as I'm describing it here. Abstractions aren't perfect. It is often necessary for performance or debugging purposes to have that deeper understanding, but that doesn't mean every programmer needs that understanding, or needs it all the time.

> The business developer within me knows that applications have to be completed and put into productive use more quickly than ever

The software development profession is made up of many different types of skills. An education in classical computer science is a strict requirement for only a few of those types. There are many people who contribute a great deal to the profession without that piece of parchment. Ultimately, I would rather have an inclusive rather than exclusive profession. Many people who are not educated as computer scientists have a great deal to contribute to the profession. Depending on their inclination and effort, they could even have as good a formal grounding as a computer science graduate. To claim that the parchment is necessary is dubious, but to claim that the knowledge is necessary is not.

Take a Seat

Yet there is also ample room for those who write code fast, without necessarily having a deep understanding of the execution model. What such programmers lack, they make up for with productive use of modern tools, and there remains a role for programmers who can "carry the wood" by delivering working applications quickly in response to business needs.

There are perhaps more significant arguments on whether those lacking a classical computer science education and skills are devaluing their labor. Petzold comes right out and makes that claim. Spolsky is a little more circumspect, noting only that he can no longer distinguish between an average programmer and a great one. That question has some relevance in an era where the value of a contribution can determine its geographic location.

However, I still think it's the wrong question. As individuals we have a great many issues to consider in building our skills base and career. As long as we have a foundation for lifelong learning, we can choose to use the tools that make best use of those skills. Anyone who banks a software career on rote expertise with one or two specific tools will be in a different career before too long.

Which is more important, fast code or good code? Delivering the application, or building an application to last? The answer can only be that where you stand depends on where you sit. Those charged with delivering applications in response to business needs must out of necessity make full use of the most productive tools. It would be ideal for these programmers to be able to write lasting code at the same time, but compromises and trade-offs are the hallmark of computing.

After all of the advances in tools and software engineering practices, it still comes down to the old saying: "cost, schedule, or quality; pick any two." The nature of the applications has changed, and the need for software has increased greatly, but no one has found a way to undo the trade-offs that any programmer and development team must make among these three characteristics. JP

Peter Varhol is a senior member of the technical staff for Progress Software. Contact Peter at **peterv@mv.mv.com**.

Select and Load JDBC Drivers

by Kevin **JONES**

Put your plug-in to work selecting and loading JDBC drivers to support a variety of databases

In a previous installment we set up a plug-in to display JDBC requests by including the JDBC driver's JAR files in the plug-in JAR file. Doing so obviously raises an issue because one of the aims of the plug-in is to be generic. Here we will examine how to load JDBC drivers and how to use Standard Widget Toolkit (SWT) to select the JAR files containing those drivers.

JDBC drivers are contained in JAR files. For example, the Microsoft SQL driver classes that I've been using so far are contained in three JARs: msbase.jar, mssqlserver.jar, and msutil.jar. Currently these files are included in the plug-in's JAR file and therefore on the plug-in's classpath. For the plug-in to be useful we need to be able to dynamically load the driver code. The first part of this challenge is to allow the user to select the JARs that will later be added to the classpath.

We start by adding another section to the plug-in's user interface (UI) to list the JARs that we'll use. The UI will contain three buttons: one to add JAR files, one to remove a single JAR file, and one to remove all JAR files (see Figure 1). The Remove and Remove All button implementations are straightforward; we won't spend time on those implementations here (of course, all of the code is available at www.javapro.com, where you'll find those implementations). Instead, let's look at the Add implementation.

Standard Opening

The code for the Add widgetSelected handler uses the SWT FileDialog class to display a dialog box for file selection. This dialog will use the operating system's underlying Open dialog and fits in with the standard user experience. To create the FileDialog you need to provide a couple of parameters: the shell the dialog belongs to and a set of flags to control the dialog's UI. The code looks like this:

```
// _add is a reference to the
// 'Add' button in the panel
_add.addSelectionListener(
   new SelectionListener()
{
   public void widgetSelected(
      SelectionEvent arg0)
   {
      FileDialog fileDialog =
         new FileDialog(
            _mainPanel.getShell(),
            SWT.OPEN | SWT.MULTI);
```

This code will create an Open (SWT.OPEN) dialog parented by Eclipse. It will also allow you to choose multiple files (SWT.MULTI), which is exactly what we need. Once we have the dialog object, we want to display it, but before we do that we have to tell the dialog which files to display. The dialog has two methods: setFilterNames() and setFilterExtensions(). The setFilterExtensions() method is used by the dialog to filter the files being displayed; the setFilterNames() method is used by the dialog to allow the user to choose the filter the dialog will use. We call the methods this way:

```
fileDialog.setFilterNames(
   new String[] {
   "JAR Files (*.jar)",
   "Zip Files (*.zip)",
   "All Files (*.*" });

fileDialog.setFilterExtensions(
   new String[] { "*.jar",
   "*.zip",
   "*.*" });
```

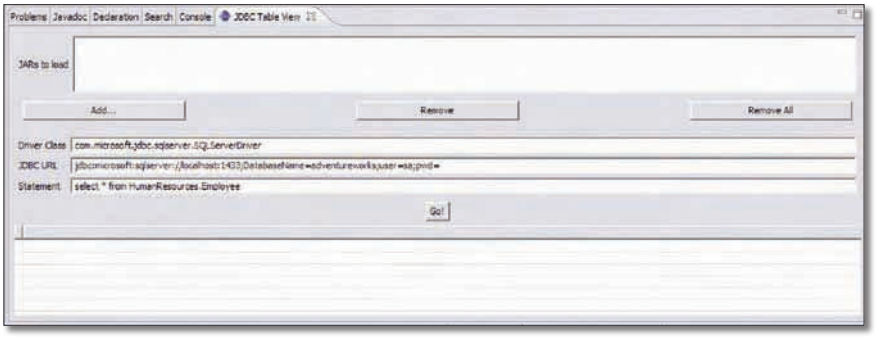


Figure 1 | Adding JARs The UI contains three buttons for managing JAR files: Add, Remove, and Remove All. We'll focus on the Add implementation.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0601 Download all the code for this issue.

JP0601PI Download the code for this article separately. The code demonstrates building a JDBC Viewer plug-in for the Eclipse platform.

Read More

JP0601PI_T Read this article online.

JP0511PI_T Read the related article "Put a Plug-In to Use" by Kevin Jones.

JP0507PI_T Read the related article "Take a JFace Detour" by Kevin Jones.

You can see how these two methods work together to manage the display of the dialog. We can show the dialog as it appears in Windows (see Figure 2) like this:

```
if (fileDialog.open() != null)
{
```

Notice that the dialog is a standard Windows' Open dialog that allows the user to select the correct files. Once the user selects the files two things need to happen: the files have to be added to the list box on the plug-in's UI, and the file names have to be passed to the code used to load the JDBC driver. To do both of these tasks you need to get a list of the files selected. The FileDialog returns the list of selected files through the getFileNames() method. After calling this method we add the files to the list box. The List widget is called _jars:

```
String[] fileNames =
   fileDialog.getFileNames();
for (String fileName : fileNames)
{
   StringBuffer sb =
      new StringBuffer(
      fileDialog.getFilterPath());

   if (sb.charAt(sb.length() - 1)
      != File.separatorChar)
      sb.append(File.separatorChar);

   sb.append(fileName);
   _jars.add(sb.toString());
```

This code gets the file name from the String array returned by getFileNames(), and from this method it has to create a fully qualified file name. The path to the file is obtained by calling getFilterPath() on the FileDialog, and we then check to see if this path ends in the system-defined path separator (\ or /); if not, add the separator before appending the file name. The fully qualified file name is then added to the list.

Now we need to pass these file names to the model by amending the JDBCTableModel constructor to accept an array of file names and then add these files to the classpath. The model has to check that the URLs are all valid. To do this check provide a helper method called LoadLibraries() that takes an array of Strings, with each String being a file name.

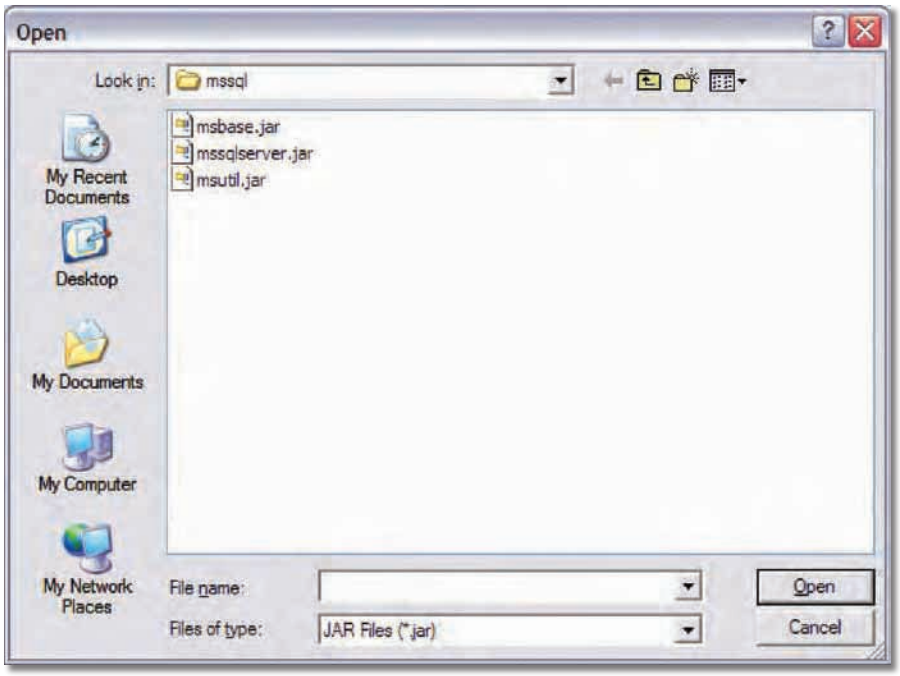


Figure 2 | Dialog Filter The setFilterNames() and setFilterExtensions() methods work together to manage the display of the Open dialog.

Valid Connections

For each String in the array LoadLibraries() creates a java.net.URL object and attempts to connect to the URL by calling openConnection() followed by getInputStream(). If both of these calls succeed then the URL is valid, and we add the URL to a list. Finally the method converts the list of URLs to a URL array and returns this array. The JDBCTableModel constructor calls the LoadLibraries() helper method, passing the array of file names (see Listing 1).

Once the constructor has the array of URLs representing the JARs containing the classes needed by the JDBC driver, it can now load the driver. However, this process is not quite as simple as it should be. To load a JDBC driver you typically use (assuming you are not using JNDI):

```
Driver driver = Class.forName(
   JDBCDriverClass);
Connection conn =
   DriverManager.
   getConnection(url);
```

However, it isn't going to work in this instance for a couple of reasons. As the driver class is not on the classpath, Class.forName() will not find it. This condition is easy to fix. The Class.forName() method accepts a classloader, which it will use to find the class to be loaded. The code will look something like this:

```
URLClassLoader ucl =
   new URLClassLoader(urls);
   Driver driver = (
   Driver)Class.forName(
   driverClass, true, ucl);
Connection conn = DriverManager.
   getConnection(url);
```

We create the ClassLoader—in fact, a URLClassloader—by passing the constructor an array of URLs. These are the URLs we just created. However, if you try this code you will find that it fails, and the reason for the failure is documented in the help for DriverManager:

When the method getConnection is called, the DriverManager will attempt to locate a suitable driver from amongst those loaded at initialization and those loaded explicitly using the same classloader as the current applet or application.

Since we are loading the classes dynamically we are not using either of these two classloaders. To work around this failure we have to make the DriverManager believe we are loading the driver from one of the two classpaths mentioned previously, and one way of doing that is to create a *shim* class. The shim class is a very thin veneer

Listing 1 URL Directory

```
private URL[] LoadLibraries(String[] files)
{
  java.util.List<URL> urlList = new ArrayList<URL>();
  for (int i = 0; i < files.length; i++)
  {
    URL u;
    InputStream is = null;
    try
    {
      u = new URL("file", null, "/" + files[i]);
      URLConnection uc = u.openConnection();
      is = uc.getInputStream();
      urlList.add(u);
    }
    catch (MalformedURLException e)
    {
      JdbcviewerPlugin.log.log(new Status(
        Status.ERROR,
        "com.develop.kevin.jdbcviewer", 1,
        "Error Loading JDBC Library", e));
    }
    catch (IOException e)
    {
      JdbcviewerPlugin.log.log(new Status(
        Status.ERROR, "com.develop.kevin.jdbcviewer",
        1, "Error Loading JDBC Library", e));
    }
    finally
    {
      try
      {
        is.close();
      }
      catch (IOException e){}
    }
  }
  return (URL[]) urlList.toArray(new URL[0]);
}
```

The LoadLibraries() method converts the list of URLs to a URL array and returns it.

Listing 2 Shimming an Interface

```
import java.sql.Connection;
import java.sql.Driver;
import java.sql.DriverPropertyInfo;
import java.sql.SQLException;
import java.util.Properties;

public class DriverShim implements Driver
{
  private Driver driver;

  public DriverShim(Driver d)
  {
    this.driver = d;
  }

  public boolean acceptsURL(
    String u) throws SQLException
  {
    return this.driver.acceptsURL(u);
  }

  public Connection connect(
    String u, Properties p) throws SQLException
  {
    return this.driver.connect(u, p);
  }

  public int getMajorVersion()
  {
    return this.driver.getMajorVersion();
  }

  public int getMinorVersion()
  {
    return this.driver.getMinorVersion();
  }

  public DriverPropertyInfo[] getPropertyInfo(
    String u, Properties p) throws SQLException
  {
    return this.driver.getPropertyInfo(u, p);
  }

  public boolean jdbcCompliant()
  {
    return this.driver.jdbcCompliant();
  }
}
```

To make the DriverManager believe we're loading the driver from one of two classpaths, we create a shim class that implements the Driver interface and passes all calls on the real driver.

over a real driver. It implements the Driver interface and simply passes all calls on the real driver. A reference to the real driver is passed to the shim's constructor. The shim class is provided as part of the plug-in code and will be loaded by the classloader that loads the plug-in. It can be used by the DriverManager (see Listing 2).

To use this shim we have to create an instance of our real driver, and then register the shim with the driver manager by passing a reference to the real driver to the shim. The code looks like this:

```
URLClassLoader ucl =
   new URLClassLoader(
   urls);Driver driver =
   Driver driver = (
      Driver)Class.forName(
      driverClass, true,
      ucl).newInstance();

DriverManager.registerDriver(
      new DriverShim(driver));

conn = DriverManager.
      getConnection(connectionString);
```

Getting There

Notice that we create an instance of our driver by calling newInstance() on the loaded class. We then explicitly register the DriverShim with the DriverManager (something we wouldn't do normally). However, after that we use the driver as normal.

The plug-in is now more complete. We can load whatever JDBC drivers we need to work with a particular database. However, some things are still missing. For example, we should not have to hard code the JDBC connection strings and the other text that we use but instead should be able to persist that after the end of each session. Eclipse supports that, and we'll see how to use the persistence mechanism in a future column. *JP*

Kevin Jones is a developer who researches and teaches Java programming and explores HTTP and XML. He lives in the U.K. and works for Developmentor, a training company based in the United States and Europe that specializes in technical training on Java and Microsoft platforms. Contact Kevin at **kevinj@develop.com**.

Free Article Archives

For FTPOnline Members

Thousands of articles and code samples are available from our library of FTP magazines: *Visual Studio Magazine/Visual Basic Programmer's Journal, Windows Server System Magazine/.NET Magazine, Enterprise Architect, Java Pro,* and *XML & Web Services Magazine.*

The original just keeps getting better. Join at: www.ftponline.com/members Register today!

www.ftponline.com/archives

FTPOnline

Hunting the Unicorn

A little trickery overcomes adapting diverse object classes to a mutual framework when building custom tools

by Daniel F. **SAVARESE**

Over time, past-conquered programming challenges have a habit of reincarnating themselves. They look slightly different, yet underneath their skins beat the hearts of beasts felled previously by tried and true daggers of design. You recognize the weapons you must bring to bear as you rejoin a battle best avoided. Combating the fiends consumes precious time even though your victory is certain.

Patterns of design are the weapons we apply to recurring programming challenges, and reimplementing those patterns in similar situations is the combat we endure. By implementing generic code that can be reused in many situations, we can dispatch our enemies quickly or avoid combat altogether. I liken the quest for a single implementation that can be applied in many instances to a hunt for the elusive unicorn. The animal may be subdued only through trickery. Once captured, its horn will cure all poisons.

I've been troubled of late by the little tools I build to perform quick tests or aid in debugging. It seems I build the same sorts of tools for different projects, but can't use the same code because of implementation-level differences. The functions of the tools are identical, yet I have to spin cycles recreating them. With the aid of open source software libraries, this sort of tool building takes far less time than it might otherwise. For example, I've used the Jakarta Commons BeanUtils library many times to convert text to class instances and back again. The practical goal of reusable software is not elegance of design, but the saving of time.

Nevertheless, it takes time to save time. You've got to put in the hours to figure out how to generalize the solution to one problem so that it can be applied to many related problems. When you build a simple development aid quickly, you don't take the time to generalize its implementation. Your primary objective is to get back to writing the main project's software, not to dilly dally with development support software. That's why the same tools have to be built time and again for new projects.

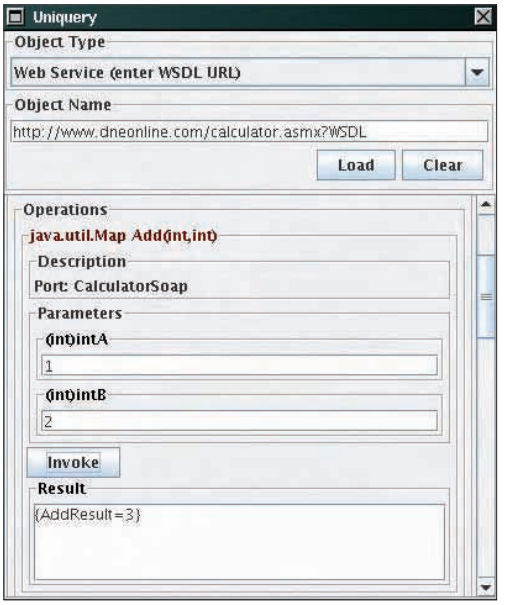


Figure 1 | Discovery Uniquery uses the ObjectInspectorPanel to discover and invoke the operations of Web services, JavaBeans, and MBeans.

Enough, Already

With that in mind, I'd like to take the time to step through the process of generalizing a development aid I've had to implement two times. When the situation cropped up a third time, I said, "enough is enough!" and decided to implement the tool in a generic and reusable fashion. A couple of years ago I was writing self-reconfiguring middleware. The system was built using Java Management Extensions (JMX) MBeans to facilitate introspection and external management. As part of the development process, I needed to be able to examine and modify live configurations, individual object instances, and other run-time state. To do this work, I wrote a simple configuration tool that included a component that could dynamically assemble a graphical user interface (GUI) exposing the operations and attributes of an MBean.

Lo and behold, not long afterward, I had the need for a similar tool for working with JavaBeans. The implementation was nearly identical, but it relied on the BeanInfo class and the java.beans package instead of MBeanInfo and the javax.management package. I discussed a simple version of this tool in a previous column

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0601 Download all the code for this issue.

Read More

JP0601PS_T Read this article online.

JP0407PS_T Read the related article "Introspective JavaBeans" by Daniel F. Savarese.

EA0402AT_T Read the related article "Component Asset Management" by Lee Sherman.

JP040421PC_T Read the related article "Awaken the Service Locator" by Paulo Caroli.

Listing 1 Resolve Impedance Mismatch

```
package org.savarese.unicorn.ui;

import javax.management.MBeanInfo;

public interface ObjectPanelModel<O extends Object> {
  public void setObject(O obj);
  public O getObject();
  public String getObjectName();
  public MBeanInfo getObjectInfo();
  public boolean isValid();
  public Object getAttribute(String attribute)
    throws Exception;
  public void setAttribute(String attribute,
    Object value) throws Exception;
  public Object invoke(String operation,
  Object[] params, Class[] types, String[] signature)
    throws Exception;
  public void addObjectPanelModelListener(
    ObjectPanelModelListener<O> listener);
  public void removeObjectPanelModelListener(
    ObjectPanelModelListener<O> listener);
}
```

We resolve differences in object representations by hiding them behind the ObjectPanelModel interface.

Listing 2 Taming the Beast

```
package example.uniquery;

import java.io.IOException;
import java.util.Map;
import javax.swing.*;
import javax.management.*;
import javax.management.remote.*;
import javax.wsdl.*;

import org.apache.wsif.util.*;

import org.savarese.unicorn.ui.*;

public final class Main {
  public static final class JavaBeanPanelModelFactory
    implements ObjectPanelModelFactory
  {
    public ObjectPanelModel createObjectPanelModel(
      String objectName)
      throws ClassNotFoundException,
      InstantiationException,
      IllegalAccessException
    {
      return new JavaBeanPanelModel(Class.forName(
        objectName).newInstance());
    }
  }

  public static final class
    WebServicePanelModelFactory
    implements ObjectPanelModelFactory
  {
    public ObjectPanelModel createObjectPanelModel(
      String objectName)
      throws WSDLException
    {
      Definition wsdl = WSIFUtils.readWSDL(
        null, objectName);
      Map services = wsdl.getServices();

      // return first service found
      for(Object service : services.values())
        return new WebServicePanelModel(
        wsdl, (Service)service);

      return null;
    }
```

Continued on page 38

Creating a series of ObjectPanelModelFactory implmentations executes the Uniquery tool (see Figure 1). Add a new object type by implementing a new factory.

("Introspective JavaBeans," *Java Pro*, July 2004). Most recently, I needed to dynamically construct a user interface for accessing arbitrary Web services based on a WSDL descriptor. That's when I decided to stop and build a small framework to simplify this sort of dynamic inspection of objects. Even though each tool had slightly different requirements, they all needed a Swing component that could discover and expose the public operations and attributes of an object.

Before continuing, I'll mention that the code for the Swing component, which I've named ObjectPanel, is very similar to that for BeanPanel in my earlier column. Therefore, I will present only the key abstraction (see Listing 1) for the generic version of the software and an example (see Listing 2) of how to use that abstraction to build a generic tool (see Figure 1). (See Resources online at www.javapro.com for the complete source code download under the Apache License 2.0).

> Assembling a GUI for interacting with an object is straightforward once you have a representation for that object

Assembling a GUI for interacting with an object is straightforward once you have a representation for that object. The difficulty in implementing a GUI for interacting with different classes of objects—such as MBeans, JavaBeans, and Web services—lies not in the GUI code, but in adapting each to a common representation structure. In the J2SE core API and the Java APIs for WSDL, we have javax.management.MBeanInfo to represent MBeans, java.beans.BeanInfo to represent JavaBeans, and javax.wsdl.Service to represent a Web service. All of these representations have to be resolved into a single form that can be manipulated in the process of assembling a GUI.

Not only do we need to be able to display information about an object, but also

Continued from page 37.

```
  }

  public static final class MBeanPanelModelFactory
    implements ObjectPanelModelFactory
  {
    private JMXConnector connector = null;

    // Expects URLs of form:
    // JMXServiceURL?ObjectName
    // e.g.,
    // service:jmx:jmxmp://localhost:8082?domain:name=
    // Status,instance=2
    public ObjectPanelModel createObjectPanelModel(
      String objectName)
      throws Exception
    {
      close();

      String serviceURL = objectName.substring(
        0, objectName.indexOf('?'));
      String mbeanName = objectName.substring(
        objectName.indexOf('?') + 1);

      connector =
        JMXConnectorFactory.newJMXConnector(
          new JMXServiceURL(serviceURL), null);
      connector.connect();

      return new MBeanPanelModel(
        new ObjectName(mbeanName),
        connector.getMBeanServerConnection());
    }

    public void close() throws IOException {
      if(connector != null)
        connector.close();
    }
  }

  public static final void main(
    String[] args) throws Exception {
    ObjectInspectorPanel inspector =
      new ObjectInspectorPanel();
    inspector.addObjectPanelModelFactory(
      "JavaBean (enter class name)",
      new JavaBeanPanelModelFactory());
    inspector.addObjectPanelModelFactory(
      "Web Service (enter WSDL URL)",
      new WebServicePanelModelFactory());
    final MBeanPanelModelFactory mbeanFactory =
      new MBeanPanelModelFactory();

    inspector.addObjectPanelModelFactory(
      "MBean (enter JMXServiceURL?ObjectName)",
      mbeanFactory);

    Runtime.getRuntime().addShutdownHook(
      new Thread() {
        public void run() {
          try {
            mbeanFactory.close();
          } catch(IOException ioe) {
            ioe.printStackTrace();
          }
        }
      });

    JFrame frame = new JFrame("Object Inspector");
    frame.setDefaultCloseOperation(
      JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(inspector);
    frame.setSize(400, 400);
    frame.setVisible(true);
  }
}
```

we need to be able to invoke the object's operations. The javax.management.remote package provides for setting the attributes and invoking the operations of remote MBeans. The java.beans package relies on the classes in java.lang.reflect to modify JavaBeans. The Java APIs for WSDL do not provide for invoking Web services. We need to use JAX-RPC or Java API for XML Messaging (JAXM) for that.

Now You See It...

These disparate means for representing and executing objects appear, at first glance, to be impossible to reconcile. Not to worry; we can use trickery to capture our unicorn. We can reuse existing software as much as possible. Instead of inventing a framework of adapter classes to model objects, we can use JMX MBeanInfo and its associated classes. The JMX designers already tackled the problem of modeling disparate objects. So why not benefit from their success? Instead of writing a framework of classes that bridge the WSDL APIs with JAX-RPC and JAXM, we can use the Apache Web Services Invocation Framework (WSIF). Not only does WSIF allow us to generate Java classes to invoke Web services, but also it allows us to dynamically invoke services without creating stub classes.

By adopting the JMX API as our object representation framework, we must write a converter from each alien representation to MBeanInfo. Converting from BeanInfo to MBeanInfo does not require much effort because both frameworks share a similar structure. For example, MBeanOperationInfo corresponds to MethodDescriptor, MBeanAttributeInfo corresponds to PropertyDescriptor, and so on.

The WSDL APIs exhibit a similar correspondence. We could have taken a different approach and used the WSDL service model instead of the JMX model. The Web Services Invocation Framework (WSIF) can invoke not only Web services based on their WSDL descriptions, but also JavaBeans and any other object type for which you can write a WSDL description. However, for our purpose, the approach of using WSDL descriptors seems less efficient and requires the additional step of writing WSDL descriptions of objects.

These disparate means for representing and executing objects appear, at first glance, to be impossible to reconcile. Not to worry; we can use trickery to capture our unicorn

Now that we have settled the object representation problem, we are left with the invocation problem. Each object sys-

tem may have a different set of requirements to allow invocation. For example, MBeans require an MBeanServerConnection. The best we can do is satisfy the needs of the systems with which we are familiar. At a later point in time we may have to adjust our framework to meet the needs of a new system. In the meantime, we hide object representation and invocation details behind the ObjectPanelModel interface (see Listing 1). No matter what object type we are working with, as long as we implement an ObjectPanelModel for it, our ObjectPanel component will be able to display and interact with the object.

The parameters of the invoke() method represent a compromise for efficiency. MBeans expect method signatures expressed as strings, and JavaBeans expect signatures in terms of class objects. Instead of requiring an ObjectPanelModel implementation to convert from one to another on each method invocation, we supply both forms. It so happens that ObjectPanel has already obtained both representations for its own use. Therefore, we can avoid a redundant calculation.

Having already implemented ObjectPanelModels for MBeans, JavaBeans, and Web services, the task of building a generic development aid is reduced to the less than an hour of programming it took to write the Uniquery example (see Listing 2). The code is so short because we've already invested the time in creating the ObjectInspectorPanel class (see Figure 1). To build tools that use the inspector, we only have to populate it with model factories. Note, however, that the tools themselves need to be cognizant of the details of a specific ObjectPanelModel implementation. For example, Uniquery knows an MBeanPanelModel requires an MBeanServerConnection and takes care to close the connection when it is finished using it. It must also split a custom MBean identifier into a JMX service URL and ObjectName.

Generic Mindset

If I had invested the time to build a generic object inspector the first time I needed one, I would have spent less time in total than I did by implementing two application-specific object inspectors and one generic inspector. Now that I have the generic inspector, I can apply it in a new situation by implementing an ObjectPanelModel and an MBeanInfo converter. Writing two classes takes considerably less time than would be consumed otherwise. When building custom tools for a development project, it can pay to think generically instead of specifically. JP

Daniel F. Savarese is an independent software developer and technology advisor. He founded ORO Inc., was a senior scientist at Caltech's Center for Advanced Computing Research, and was vice president of software development at WebOS. Daniel wrote the original Jakarta ORO, Commons Net, RockSaw, Sava Algorithms, and Unicorn libraries. He also coauthored *How to Build a Beowulf* (MIT Press, 1999) and earned a Ph.D. in computer science from the University of Maryland College Park. Contact Daniel at **www.savarese.org/contact.html**.

Java's Innovation Engine

Guest Opinion
by Mike **MILINKOVICH**

The trend is clear, open source communities such as Apache and Eclipse are driving the innovation taking place in Java today. And this is very good news for all of us who care deeply about the future of Java.

From its inception, the Java Community Process (JCP) was developed to bring order and process to the development of standard APIs across Java. The focus at Sun Microsystems was to ensure a cohesive intellectual property policy, API compliance testing, and a controlling interest in all Java-branded technology. By all measures it has been a success, and the current mainstream success enjoyed by Java is owed in no small measure to the successful execution of the JCP process.

However, this success came at a cost: stifled innovation. It should be no surprise that standards-setting bodies are not the place for developers to push the envelope. Great ideas are neither created nor evangelized by committees, no matter how expert. The result? Compare and contrast the utility of Enterprise JavaBeans (EJB) 1.0 entity beans versus Hibernate, or the EJB container versus Spring to name just two.

The reason why open source projects are innovation engines is inherent in the processes used by those communities. The transparency of open source requires that both ideas and their tangible implementations are exposed to the never-ending scrutiny and criticism of a broad-based cohort of developers attempting to build and deploy working systems. It is Darwinism at its finest, where weak ideas and shoddy implementations are quickly relegated to the CVS attics of the world. But just as importantly, open source is where great ideas implemented gracefully inspire and excite developers to use and evolve them quickly.

Another important facet of great ideas is that they inspire controversy. One of my favorite blogs is "Creating Passionate Users" (http://headrush.typepad.com/), where Kathy Sierra regularly makes the point that if a new idea is met with consensus approval, it's probably uninteresting. Great ideas and innovations change things, and change inevitably meets with resistance. In other words, if you're not at least occasionally pissing people off, you're probably not trying.

What are some of the key Java innovations that were born in open source? There are too many to list in the space here, but a small sampling of my personal favorites would include Struts, Spring, Hibernate, Matisse, and the Standard Widget Toolkit (SWT).

Apache Struts delivered a working Model-View-Controller (MVC) pattern implementation to Java Web development. The Spring framework is most popular among the lightweight containers, and it allows developers to focus on the business logic rather than expending effort on working with containerisms. Hibernate is the practical alternative to EJB persistence, and it allows developers to develop complex applications with persistent POJOs.

By all reports, Matisse is a slick GUI builder for Java that introduces its own LayoutManager (GroupLayout) that is distinct from those provided by the JDK. An interesting illustration of this article's premise is that even Sun has turned to its NetBeans open source project to create innovation. Probably the poster child for a great idea that inspires controversy, SWT gives Java developers the freedom to create first-class native applications on many platforms. Say what you want about the Swing versus SWT religious war. Enabling the freedom to build a type of application not previously possible with Java is a useful innovation for the platform and the community.

I am sure that there are no surprises in that list. Everyone has heard of those frameworks, and most have at least tried them out. The real point is that none of those highly useful tools could have possibly been brought into this world through a standards process. The processes of reasonable negotiation, of compromise, and of reconciling competing corporate agendas serve to smooth the rough edges and dull the brilliance of great ideas. Every one of those examples is ultimately the work of either a single person or a very small group of people striving to solve a difficult problem with style and grace.

If open source is clearly where innovation is occurring, does that mean that the JCP has no role? Not at all. There are good reasons to create standards. They provide a very important element of investment protection for the companies that adopt Java technology for their products and/or internal enterprise systems. However, in a perfect world, we would start to see the JCP consciously working with the broader Java community to attract the projects and ideas that have been clearly shown in the marketplace to be successful. Innovate first, demonstrate utility and adoption second, and then standardize.

The inclusion of many of the best ideas from Hibernate in EJB 3.0 is a demonstration of this process in action. There are signs that GroupLayout may migrate someday from Matisse to the JDK. Yet in many cases the JCP ignores the open source alternatives, or worse yet, in cases like JSR 198 (versus Eclipse) and JSR 277 (versus Apache Felix and Eclipse Equinox), views them as competitors that must be avoided—to the detriment of Java supporters everywhere. JP

Mike Milinkovich is executive director of the Eclipse Foundation Inc. (www.eclipse.com)

FTPOnline

One Source for All Your Technical Information

Newly Expanded, Easily Accessible

1 CHANNELS
To better serve your needs, FTPOnline has been restructured around seven channels: Architecture, Java, .NET Development, Windows IT, ASP.NET, Database and Security. More channels to come!

2 SPECIAL REPORTS
Get comprehensive information on subjects critical to all IT professionals, such as Security, Service-Oriented Architecture, and Operations Management.

3 WEBCASTS
Watch and listen to industry experts discuss hot IT topics.

4 WHITE PAPERS
Download white papers that examine evolving technologies.

5 RSS FEED
Get quick updates on the latest blogs and articles published at FTPOnline.

6 MAGAZINES
Filled with downloadable code, interviews with industry visionaries, in-depth tutorials, overviews of implementation and management strategies, article archives, and more!

7 NEWSLETTERS
Free e-mail newsletters in your area of interest, delivered right to your inbox.

Go there today:

www.ftponline.com

FTPOnline

Sybase® WorkSpace: Bridging the Gap Between SOA and Traditional Tooling

Sybase WorkSpace, a unified application development environment, is the first designed to bridge the gap between the vision of a service-oriented architecture (SOA) and the reality of what traditional development tools can deliver. Sybase WorkSpace combines modeling, data management, services assembly and orchestration, Java™ development and mobilization in a single tool. Designed to support the principles of service-oriented development of applications (SODA), Sybase WorkSpace enables developers to quickly build and deliver many kinds of applications: from event- and data-driven applications to web-based, composite, and mobile applications. Sybase WorkSpace is built upon the Eclipse open source framework, making it easier and faster for developers to build complex applications that leverage heterogeneous infrastructures.

Get Sybase WorkSpace and start building tomorrow's applications today.

For more information and to download white papers, visit www.sybase.com/workspace

SYBASE®